UNIVERSIDAD DE LA RIOJA

TRABAJO FIN DE ESTUDIOS

Título

Localización de objetos en imágenes mediante técnicas de aprendizaje profundo

Autor/es

LUCÍA AYESTARÁN GARRALDA

Director/es

JÓNATAN HERAS VICENTE

Facultad

Facultad de Ciencia y Tecnología

Titulación

Grado en Ingeniería Informática

Departamento

MATEMÁTICAS Y COMPUTACIÓN

Curso académico

2017-18

publicaciones.unirioja.es
E-mail: publicaciones@unirioja.es

UNIVERSIDAD DE LA RIOJA

Facultad de Ciencia y Tecnología

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Localización de objetos en imágenes mediante técnicas de aprendizaje profundo

Alumna:
Lucía Ayestarán Garralda
Tutor:
Jónathan Heras

Logroño, febrero de 2018

ÍNDICE

RESUMEN

La inteligencia artificial está cada día más presente en las nuevas tecnologías gracias, en gran medida, al éxito de las técnicas de aprendizaje profundo. Entre las áreas que han tenido más impacto en el aprendizaje profundo se encuentra la visión por computador, encargada de comprender las imágenes del mundo real con el fin de producir información tratable por un ordenador. Uno de los problemas que aborda esta disciplina consiste en localizar objetos concretos en imágenes (es decir, encontrar la posición del objeto dentro de la imagen), y este es el problema que motiva este trabajo. En concreto, trataremos de crear un programa que, utilizando técnicas de aprendizaje profundo, sea capaz de determinar la posición de un violín en una imagen, con el fin de que dicho programa pueda ser adaptado y reutilizado en futuros proyectos del grupo de informática de la Universidad de La Rioja.

Sin embargo, la misma ciencia que parece ofrecer la solución a tantos problemas no es, ni mucho menos, fácil de implementar debido a que los algoritmos son muy recientes y no están excesivamente documentados. Además, no existe una única vía y, en muchas ocasiones, es difícil acertar a la primera con la mejor solución para un problema concreto. Por ello, en este trabajo analizaremos las opciones que el aprendizaje profundo nos ofrece para resolver el problema de la localización, y trataremos todas las fases necesarias para construir un problema de localización, desde la captura y anotación de imágenes hasta la evaluación de los algoritmos.

ABSTRACT

The world of artificial intelligence is more and more present in new tecnologies, and the reason is highly related to the success of Deep Learning techniques. Computer vision is one of the most popular areas where Deep Learning has been successfully applied, and its goal is to treat real world's images to be able to produce information that a computer can understand. One of the main problems studied in this discipline consists in localizing certain objects on images (that is, finding the position of an object in an image), and this is the problem that gives rise to this project. More specifically, we will try to create a program that can determine the position of a violin on an image using Deep Learning techniques. Such a program could be modified and reused on future projects of the computer science group of the Universidad de La Rioja.

Nevertheless, even if this science seems to be the solution to many problems, it is not easy to implement it at all. The reason is that it uses algorithms that are so recent and, also, little documented. Besides, sometimes, there is not a unique way approach, and rarely we are able to find the best solution for our problem in the first try. Thus, in this project, we will analize the options that Deep Learning offers for solving the localization problem, and we will work on all the different phases for setting up our solution, from image capture and annotation to the evaluation of algorithms.

INTRODUCCIÓN

Dedicaremos esta primera breve sección a la introducción del proyecto que da vida a esta memoria, explicando el contexto en el que surge y comentando algunos de los conceptos más básicos para la comprensión del problema que se nos plantea.

JUSTIFICACIÓN DEL PROYECTO

El Departamento de Matemáticas y Computación de la Universidad de La Rioja, concretamente el grupo de informática, cuenta con personal que trabaja en proyectos de análisis de imágenes biomédicas: imágenes de neuronas en enfermedades neurodegenerativas [1], de geles de ADN [2] o de antibióticos [3]. Hasta hace poco, el grupo utilizaba sobre todo técnicas de topología algebraica; sin embargo, recientemente han dado el salto a técnicas de inteligencia artificial, y en concreto de visión por computador y aprendizaje automático. Es aquí donde cobra sentido este proyecto.

Uno de los problemas que estudian en el grupo es el de la localización de objetos. Para abordarlo, en los últimos años han surgido nuevas técnicas, y en concreto surge el *Deep Learning* (o *Aprendizaje Profundo*) [4]. Así, el objetivo de este trabajo es servir como base para el grupo, de cara a aplicar este tipo de técnicas en sus proyectos. Como *proof of concept*[1] queremos ser capaces de localizar la posición de un violín en una imagen.

Este no es un trabajo estrictamente práctico, su objetivo no se reduce solo a la implementación de un programa que resuelva el problema de localización de violines, ya que pretende, además, analizar las herramientas de las que disponemos en las diferentes etapas de dicha implementación. Así, se analizarán los diferentes programas de anotación de imágenes y los modelos[2] disponibles hoy en día para resolver el problema de la localización.

CLASIFICACIÓN, LOCALIZACIÓN, DETECCIÓN Y SEGMENTACIÓN

El problema de la localización al que hacemos referencia es el tratado habitualmente en la visión por ordenador, una rama de la inteligencia artificial. Es importante, antes de nada, presentarlo formalmente y distinguirlo de los otros tres problemas principales en la visión por computador: la clasificación, la detección y la segmentación [5]**.**

- **Clasificación**

La clasificación es probablemente el problema más conocido en la visión por computador. Consiste en determinar a qué categoría, de entre muchas categorías diferentes fijadas de antemano,

[1]Una prueba de concepto o PdC (del inglés **proof of concept**) es una implementación, a menudo resumida o incompleta, de un método o de una idea, realizada con el propósito de verificar que el concepto o teoría en cuestión es susceptible de ser explotada de una manera útil.

[2] En el ámbito del aprendizaje automático, un modelo es el resultado de entrenar un algoritmo para la resolución de una tarea concreta.

pertenece el objeto que aparece en una imagen. La imagen 1 muestra un ejemplo de clasificación, en el que el objeto principal de la misma es clasificado como un gato.

En los últimos años los modelos de clasificación han superado el rendimiento humano [6]. Si bien hay un montón de desafíos sobre la clasificación de imágenes, también hay abundante información escrita sobre cómo resolverlos [7, 8].

En la actualidad, la clasificación se propone como herramienta potente para el análisis de datos a gran escala [9] y, entre otras aplicaciones, es utilizada para sistemas de detección de tráfico de vídeo [10].

Imagen 1: Clasificación de un elemento en una imagen. Resultado: el objeto principal es un gato.

- **Localización**

La localización encuentra la ubicación de **un solo objeto** dentro de una imagen, como podemos ver, por ejemplo, en la imagen 2, en la que se localiza una mariposa.

La localización se puede utilizar para muchos problemas útiles de la vida real. Por ejemplo, el recorte inteligente (saber dónde recortar las imágenes en función del lugar donde se encuentra el objeto) o incluso la extracción regular de objetos para su posterior procesamiento utilizando diferentes técnicas. Se puede combinar con la clasificación para no sólo localizar el objeto sino categorizarlo en una de muchas categorías posibles.

Imagen 2: Una mariposa localizada

El problema de la localización ha sido ampliamente abordado en los últimos años, en los que la visión por computador cobra especial importancia, y se usa en multitud de proyectos reales.

En 2006, en una conferencia internacional sobre Robots y Sistemas inteligentes en China, presentaron un marco para la planificación del agarre con un brazo robot humanoide y una mano de cinco dedos. El objetivo era proporcionar al robot la capacidad de agarrar objetos en un ambiente de cocina, basada, en parte, en un sistema de cámara estéreo para la localización de objetos en tiempo real [11].

Actualmente, la localización es ya utilizada para programar coches autónomos [12] o para ayudar a robots como Kuri a encontrar su camino [13].

- **Detección**

Si unimos la localización a la clasificación (no solo localizamos el objeto, si no que decimos de qué objeto se trata) y lo hacemos repetidamente en una imagen, el resultado es la detección, como muestra la imagen 3. La detección de objetos es el problema de encontrar y clasificar un número variable de objetos de múltiples categorías en una imagen. La diferencia fundamental es la idea de "variable". En contraste con problemas como la clasificación, la salida de la detección de objetos es variable en longitud, ya que el número de objetos detectados puede cambiar de una imagen a otra.

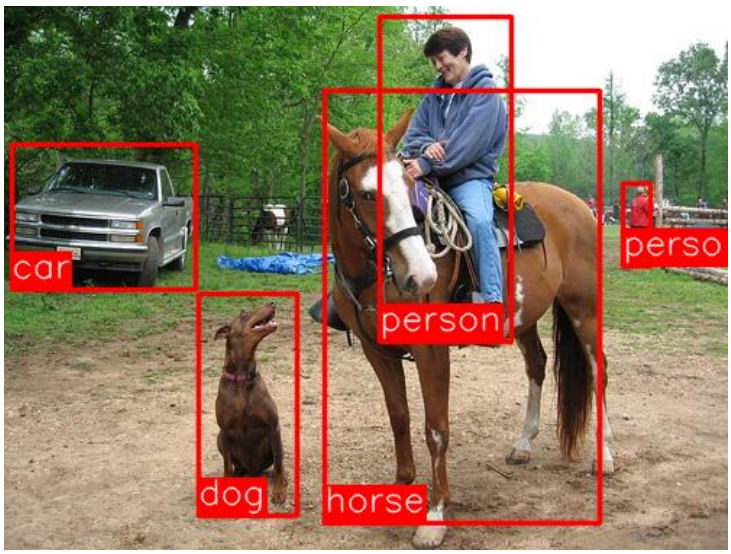


Imagen 3: Varios elementos de clases distintas localizados en una foto

Hoy en día, la detección de objetos se usa ampliamente en distintos ámbitos: por ejemplo, es utilizada para ayudar a detectar melanomas [14], y varias funcionalidades de las últimas versiones de Photoshop y Premiere Elements se basan en la detección de objetos [15].

- **Segmentación semántica**

Además de estos tres problemas, debemos introducir en esta sección el término de *segmentación semántica*, ya que, como explicaremos en la sección de Análisis, es posible que hagamos uso de ella.

La segmentación en el campo de la visión por computador consiste en dividir una imagen en varios grupos de píxeles u objetos, asignando una etiqueta a cada píxel de forma que los píxeles que

compartan la misma etiqueta también tendrán ciertas características visuales similares. Se utiliza tanto para localizar objetos como para encontrar los límites de estos dentro de una imagen.

La segmentación semántica requiere tanto segmentación como la clasificación de los objetos segmentados. Es decir, trata de entender **qué** hay en una imagen. Podemos ver el resultado de aplicar la segmentación semántica a una imagen concreta en la imagen 4.

Imagen 4: Interpretación de una imagen tras aplicar la segmentación semántica y detectar los objetos o elementos presentes en ella.

Una vez vista la diferencia entre los cuatro problemas principales de la visión por computador, podemos decir que nuestro problema encaja con el de la localización, ya que determinaremos la posición de un único violín en una imagen. Sin embargo, también estudiaremos cómo la clasificación y la segmentación semántica pueden usarse para resolver el problema de la localización.

PLANIFICACIÓN

La planificación de un proyecto es una de las tareas más importantes que hay que realizar antes de emprenderlo. Esta sección la dedicamos a exponer la metodología escogida para el desarrollo del proyecto, la Estructura de Descomposición del Trabajo (EDT) y su diccionario, el diagrama de Gantt, el calendario del proyecto junto con los hitos más importantes, y el plan de riesgos que seguiremos.

METODOLOGÍA A SEGUIR

Tras analizar las distintas metodologías existentes para este tipo de proyectos, se ha decidido utilizar la metodología en **cascada** e **iterativa**. El motivo principal es que se trata de un proyecto muy definido, en el que los requisitos están fijos desde un inicio, pero en el que, sin embargo, no sabemos qué camino nos llevará a buenos resultados. No hay un cliente como tal, lo que reduce los riesgos y hace que esta metodología sea adecuada a nuestro caso. Así, el proyecto se dividirá en las siguientes fases, que serán llevadas a cabo una detrás de la otra, como podemos ver en la imagen 5. Además, el ciclo podrá repetirse más de una vez.

1. **Análisis**: es la fase en la que se analiza y define el problema que da vida al proyecto, definiendo los requisitos que deberemos cumplir para darlo por finalizado.
2. **Diseño**: es durante esta fase cuando se toman todas las decisiones necesarias para definir el proceso de implementación y evaluación del producto que desarrollaremos (en nuestro caso, el modelo de localización).
3. **Implementación**: en esta fase se llevan a cabo todas las decisiones tomadas en el diseño y se construye el modelo.
4. **Evaluación**: en esta fase se miden los resultados obtenidos, se estudia si cumplen los objetivos marcados, y de no ser así se analiza cómo mejorar los resultados o estudiar otras alternativas.

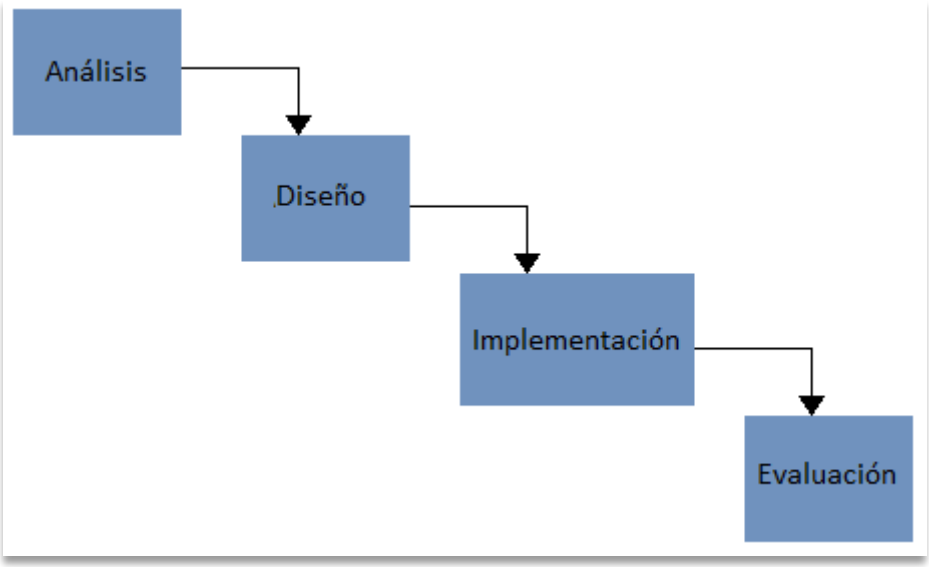


Imagen 5: Esquema en cascada.

EDT

La imagen 6 muestra la estructura de descomposición del trabajo (EDT) para el TFG. El proyecto se divide en 8 grandes bloques de trabajo, algunos de los cuales se descomponen en menores paquetes de trabajo.

DICCIONARIO DE LA EDT

La tabla 1 muestra el diccionario de la EDT, es decir, la explicación de cada uno de los nombres de los paquetes de trabajo de la misma.

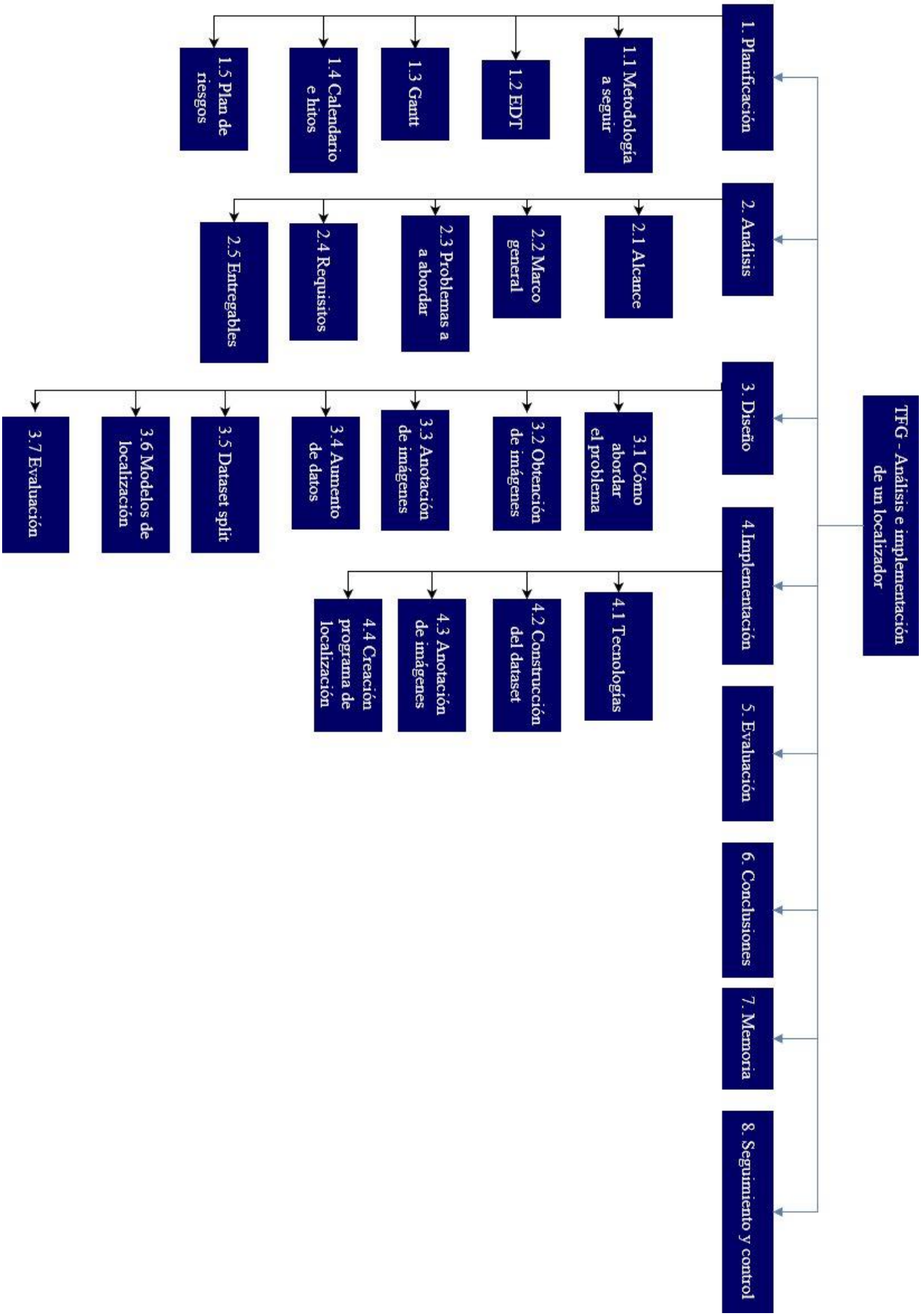


Imagen 6: Estructura de descomposición del trabajo (EDT) para el proyecto.

ID	Nombre	Descripción
1	Planificación	
1.1	Metodología a seguir	Elección de la metodología de desarrollo del TFG
1.2	EDT	Estructura de descomposición del trabajo del TFG
1.3	Gantt	Planificación y programación de tareas, estimadas en horas y duración, a lo largo del TFG
1.4	Calendario e hitos	Calendario con los meses de vida del TFG que define los hitos del TFG y su explicación
1.5	Plan de riesgos	Plan de actuación ante riesgos posibles a lo largo del proyecto
2	Análisis	
2.1	Alcance	Definición del alcance del proyecto
2.2	Marco general	Marco general en el que se desarrolla el TFG
2.3	Problemas a abordar	Requisitos hardware y software del sistema
2.4	Requisitos	Requisitos funcionales y no funcionales del trabajo
2.5	Entregables	Entregables generados en el trabajo
3	Diseño	
3.1	Cómo abordar el problema	Procesos que se llevarán a cabo para la resolución del problema planteado
3.2	Obtención de imágenes	Fuentes de captura de imágenes necesarias para la elaboración del trabajo
3.3	Anotación de imágenes	¿Qué es? Análisis de programas existentes para la anotación de imágenes y sus formatos de salida
3.4	Aumento de datos	Aplicar transformaciones a las imágenes para aumentar el dataset y mejorar resultados
3.5	Dataset split	División del conjunto de datos
3.6	Modelos de localización	¿Qué son los localizadores? Tipos. Análisis de los distintos modelos usados hoy en día para abordar problemas como el nuestro. Análisis del modelo concreto escogido
3.7	Evaluación	Cómo se evaluará el modelo. Medida de precisión
4	Implementación	
4.1	Tecnologías	Preparación del entorno
4.2	Construcción del dataset	Construcción del dataset
4.3	Anotación de imágenes	Anotación de las imágenes del dataset utilizando el programa escogido
4.4	Creación programa de localización	Implementación del programa localizador
5	Evaluación	Obtención de resultados en términos de precisión
6	Conclusiones	Extracción de conclusiones generales y personales del proyecto
7	Memoria	Realización de la memoria del TFG
8	Seguimiento y control	Seguimiento y control del cumplimiento de la planificación a lo largo del trabajo

Tabla 1: Diccionario de la EDT del proyecto.

DIAGRAMA DE GANTT

La tabla 2 muestra el diagrama de Gantt del proyecto. En él se estiman las horas de dedicación necesarias para completar cada uno de los paquetes de trabajo de la EDT y su distribución a lo largo de las 23 semanas de duración del proyecto.

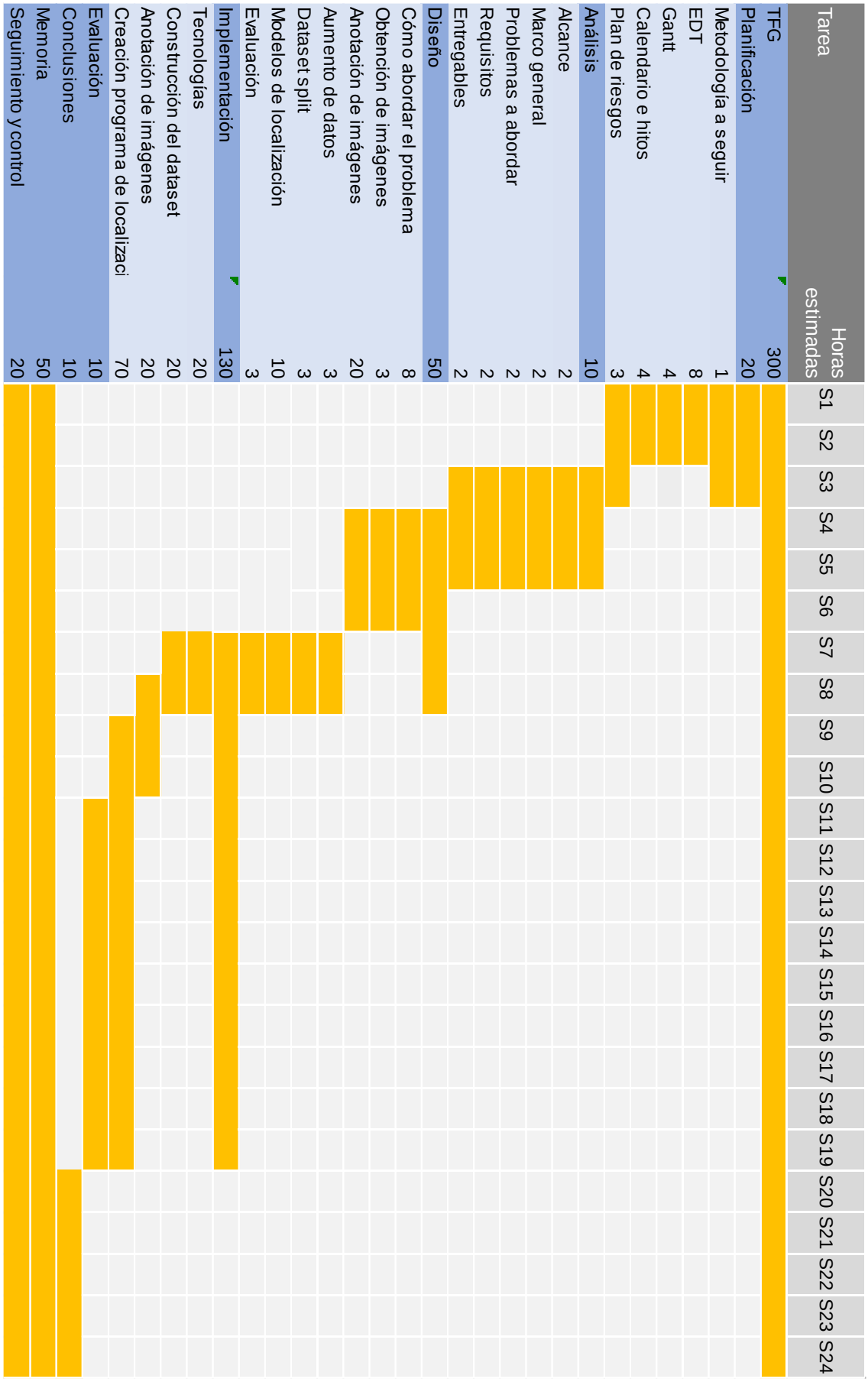


Tabla 2: Diagrama de Gantt del TFG. Horas de dedicación estimadas y en qué semanas para cada uno de los paquetes de trabajo de la EDT.

CALENDARIO E HITOS

SEPTIEMBRE							
	L	M	X	J	V	S	D
S1				7	8	9	10
S2	11	12	13	14	15	16	17
S3	18	19	20	21	22	23	24
S4	25	26	27	28	29	30	

OCTUBRE							
	L	M	X	J	V	S	D
S4							1
S5	2	3	4	5	6	7	8
S6	9	10	11	12	13	14	15
S7	16	17	18	19	20	21	22
S8	23	24	25	26	27	28	29
S9	30	31					

NOVIEMBRE							
	L	M	X	J	V	S	D
S9			1	2	3	4	5
S10	6	7	8	9	10	11	12
S11	13	14	15	16	17	18	19
S12	20	21	22	23	24	25	26
S13	27	28	29	30			

DICIEMBRE							
	L	M	X	J	V	S	D
S13					1	2	3
S14	4	5	6	7	8	9	10
S15	11	12	13	14	15	16	17
S16	18	19	20	21	22	23	24
S17	25	26	27	28	29	30	31

ENERO							
	L	M	X	J	V	S	D
S18	1	2	3	4	5	6	7
S19	8	9	10	11	12	13	14
S20	15	16	17	18	19	20	21
S21	22	23	24	25	26	27	28
S22	29	30	31				

FEBRERO							
	L	M	X	J	V	S	D
S22				1	2	3	4
S23	5	6	7	8	9	10	11
S24	12	13	14	15	16	17	18
S25	19	20	21	22	23	24	25
S26	26	27	28				

MARZO							
	L	M	X	J	V	S	D
S26				1	2	3	4
S27	5	6	7	8	9	10	11
S28	12	13	14	15	16	17	18
S29	19	20	21	22	23	24	25
S30	26	27	28	30	31		

- 7 de septiembre: Inicio del TFG
- 15 de octubre: Primer punto de control
- 15 de noviembre: Segundo punto de control
- 15 de diciembre: Tercer punto de control
- 15 de enero: Cuarto punto de control
- 18 de febrero: Fin del TFG
- Semanas 25, 26 y 27: Preparación de la defensa
- 13 de marzo: Defensa del TFG

PLAN DE RIESGOS

En esta sección definiremos la gestión tanto de las comunicaciones como de los riesgos que llevaremos a cabo a lo largo del proyecto.

- **Gestión de comunicaciones**

Nos centramos en la comunicación con el tutor porque, en este proyecto en concreto, el mismo tutor es el que ejercerá de cliente. La tabla 3 muestra la forma en la que la autora del proyecto se va a comunicar con el tutor del mismo, ya sea para transmitirle información, para pedírsela, o para llegar a un acuerdo con él. La comunicación en cada uno de los casos puede ser síncrona o asíncrona.

Tipo de comunicación	Informar	Pedir información	Llegar a acuerdo
Síncrona.	Una reunión semanal de una hora, de lunes a viernes de 15:00 a 20:00.	Reunión semanal de una hora, de lunes a viernes de 15:00 a 20:00.	Reunión semanal de una hora, de lunes a viernes de 15:00 a 20:00.
Asíncrona.	- Correo electrónico: Gmail. - A través de la aplicación del TFG.	Correo electrónico: Gmail.	Correo electrónico: Gmail.

Tabla 3: Gestión de comunicaciones con el tutor.

- **Gestión de riesgos**

Las tablas 4 a 7 muestran los riesgos existentes a lo largo del proyecto en relación a distintos aspectos del mismo. Para cada uno ellos se define un plan de actuación, que detalla qué es lo que hay que hacer para prevenir el riesgo, y qué es lo que hay que hacer en caso de que ocurra.

Riesgos con el tutor			
Riesgo	Estrategia de prevención	Estrategia de minimización	Plan de contingencia
Ausencia del tutor.	No se puede prevenir.	Tener la máxima autonomía posible.	Apoyarse en otros tutores o profesores del mismo ámbito de conocimiento.

Tabla 4: Riesgos con el tutor

Riesgos con el tamaño del producto			
Riesgo	Estrategia de prevención	Estrategia de minimización	Plan de contingencia
Falta de horas para terminar el proyecto.	Realizar una buena planificación y ajustar el alcance a la misma.	Dedicarse primero a los requisitos más importantes, dejando los demás para más adelante. Realizar un buen seguimiento y control para poder ser previsivos con el número de horas reales dedicadas a cada tarea.	Convenir con el tutor qué requisitos no se cumplirán cuando nos demos cuenta de que no podemos cumplir con el alcance. Si es demasiado tarde, ya no se puede hacer nada.
Exceso de horas planificadas no dedicadas realmente tras la realización del proyecto.	Realizar una buena planificación y ajustar el alcance a la misma. Incluir en la planificación una seria de posibles ampliaciones.	Realizar un buen seguimiento y control para poder ser previsivos con el número de horas reales dedicadas a cada tarea.	Acordar la ampliación del alcance con el tutor.

Tabla 5: Riesgos con el tamaño del producto

Riesgos con la tecnología			
Riesgo	Estrategia de prevención	Estrategia de minimización	Plan de contingencia
Mala decisión de la tecnología usada.	Realización de un buen análisis de las tecnologías más convenientes para el proyecto.	No se puede minimizar el riesgo.	Aceptar un cambio de tecnología lo antes posible sin esperar a que empeoren las cosas. En el caso de que el proyecto esté muy avanzado, no podremos hacer nada.
Fallo en la máquina virtual o la anfitriona a lo largo del proyecto.	Realización de imágenes del sistema de la máquina virtual periódicas, y almacenamiento de las mismas en un disco duro externo.	No se puede minimizar el riesgo.	Recuperación de la última imagen realizada.

Tabla 6: Riesgos con la tecnología

Riesgos con el desarrollo del producto			
Riesgo	Estrategia de prevención	Estrategia de minimización	Plan de contingencia
Pérdida de la documentación del proyecto y del código.	Realizar copias de seguridad semanales y subirlas también a la nube. Utilizar un sistema de control de versiones para el código.	No se puede minimizar el riesgo.	Recuperación de la documentación desde las copias de seguridad. Recuperación del código desde el sistema de control de versiones en la nube.

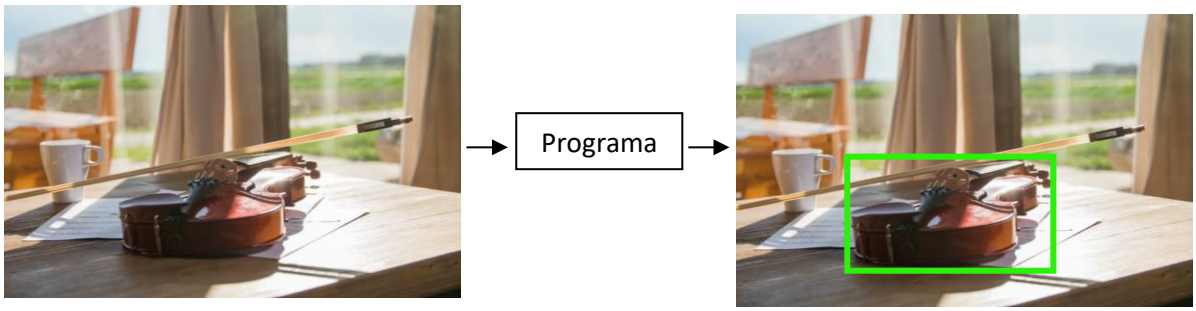
Tabla 7: Riesgos con el desarrollo del producto

ANÁLISIS

Esta sección está dedicada al análisis del proyecto, donde definiremos el alcance, el problema al que pretende dar solución, los requisitos que deberá cumplir para ello y los entregables que se generarán.

ALCANCE

El alcance de este proyecto consiste en crear un programa capaz de localizar en una imagen la posición del violín contenido en la misma, obteniendo así una imagen anotada.



ENFOCANDO NUESTRO TFG

Este proyecto surge en el grupo de informática del Departamento de Matemáticas y Computación de la Universidad de La Rioja, y pretende analizar algunas de las alternativas que surgen a la hora de abordar el problema de localización en el marco de la visión por computador. El objetivo que tiene dicho grupo con este proyecto es crear una herramienta adaptable a los problemas concretos que ellos abordan, además de conocer mejor la tecnología disponible en este momento para resolver la localización de objetos.

Dado que no sabemos cuál es el mejor método o procedimiento para solucionar el problema de la localización, nos planteamos cuatro alternativas: la primera consiste en utilizar las librerías de Deep Learning para detección de objetos, y en concreto, para la localización; la segunda aproximación se basa en utilizar modelos de clasificación ya construidos y entrenarlos para predecir los cuatro vértices del rectángulo que localiza al violín; la tercera consiste en utilizar mapas de calor a partir de modelos ya construidos; y la última se basa en utilizar la segmentación semántica. La primera es la aproximación más directa y es la más utilizada hoy en día para abordar este tipo de problemas, por lo que es la que seguiremos en un primer momento. Sin embargo, teniendo en cuenta que no sabemos la calidad de los resultados que puede producir, no descartamos las otras tres como posibles alternativas.

Para construir el modelo a partir de nuestra elección, el primer problema que se nos presenta es la elección de las librerías que vamos a utilizar. Dado que las librerías existentes para la resolución de problemas de visión por computador son difíciles de usar y analizar, y que por ello el análisis exhaustivo de las mismas se extendería tanto que escaparía del alcance de un TFG, prefijaremos

de antemano la API con la que vamos a trabajar. Así, escogeremos, guiados por el tutor, TensorFlow [16], una API libre que recientemente ha sacado Google y que es ampliamente conocida y utilizada hoy en día [17]. Solo diremos que hay otras alternativas, que, aunque no se estudiarán en detalle serán nombradas y analizadas brevemente a modo de referencia.

Además, el proceso para la implementación del programa localizador lo seguiremos de un tutorial abierto en el que se dan unas guías de uso, y que muestra cómo de general es el uso de la librería TensorFlow hoy en día [18]. Sin embargo, dicho tutorial tiene una serie de limitaciones, como, por ejemplo, que no se estudian diferentes alternativas, si no que se toma una serie de decisiones sin razonarlas.

PROBLEMAS A ABORDAR

Es normal pensar que acciones como la localización, clasificación y detección de objetos son una tarea sencilla de realizar, ya que el ser humano las lleva a cabo sin ningún esfuerzo gracias a los ojos y el cerebro. Sin embargo, el sistema que componen es altamente sofisticado y el proceso que el ser humano lleva a cabo es muy difícil de reproducir, por el momento, en máquinas. Aunque, como ya hemos mencionado anteriormente, en algunos casos se ha superado el rendimiento humano para este tipo de problemas, los recursos para conseguirlo son muy elevados y, desde luego, no al alcance de muchos.

Aunque los ordenadores están diseñados por y para los seres humanos, está claro que el ordenador es muy diferente a un cerebro humano. El cerebro humano es un sistema complejo y no lineal, y, además, su manera de procesar información es altamente paralelizada. Se basa en componentes estructurales conocidos como neuronas, que están todas diseñadas para realizar ciertos tipos de cálculos. El cerebro puede realizar una gran cantidad de tareas de reconocimiento, y normalmente las realiza en un tiempo de entre 100 y 200 ms. Tareas de este tipo siguen siendo muy difíciles para procesar en ordenadores, y hasta hace unos pocos años, la realización de estos cálculos llevaba días e incluso semanas [8].

REQUISITOS

A continuación se detallan los requisitos funcionales y no funcionales de este proyecto, así como posibles ampliaciones del mismo.

- **Requisitos funcionales**
  - Crear un programa que dada una imagen con un **único** violín devuelva las coordenadas donde este se localiza.
  - El programa podrá detectar violines en distintas imágenes en las que:
    - El violín se verá desde diferentes perspectivas
    - El violín se verá en diferente tamaño
    - El violín se verá en distintos planos (como objeto principal, en segundo plano, etc).

  - Las imágenes tendrán diferente luminosidad y serán tomadas en diferentes entornos (al aire libre, en habitaciones, en escenarios, etc).
  - Los violines tendrán características diferentes: color, tamaño, acabado…
  - Aunque únicamente habrá un violín por imagen, podrá haber objetos parecidos: violoncellos, guitarras, contrabajos…
- Utilizar la librería Tensorflow para la localización usando Deep Learning.
- El programa podrá procesar imágenes en formato PNG, JPEG y JPG.
- Construir el programa de manera que pueda hacerse uso de él mediante la línea de comandos.
- El resultado devuelto por el programa será un documento XML que contendrá la información de la localización del violín en la imagen. Además, dicha localización se visualizará automáticamente en la propia imagen.

- **Requisitos no funcionales**

- Crear un banco de imágenes con imágenes que contienen un violín.
- Anotar dicho banco de imágenes con la posición del violín en cada una de ellas.
- Estudiar el algoritmo que hay por detrás en la API de Google para la localización de objetos.
- Hacer un estudio de alternativas para anotar imágenes.
- Analizar distintos modelos para la localización de imágenes.
- Evaluar el modelo de predicción. Al ser un trabajo exploratorio no podemos saber de antemano, en principio, la precisión que debería tener. Por ello, no se exige una tasa de acierto concreta.
- Documentar el proceso de creación del programa de localización seguido con el objetivo de que sea sencillo adaptar el programa a otros problemas similares.
- Utilizar un ordenador portátil de gama media para la ejecución del proyecto, con prestaciones bastante limitadas para este tipo de problemas.

- **Ampliaciones**

- Mejorar el modelo hasta conseguir resultados de precisión por encima de un 95%.
- Comparar resultados con distintos modelos: hallar el modelo que mejor se ajusta a nuestro problema de localización en concreto.
- Crear una API para tener acceso online al localizador.
- Dar una interfaz gráfica al programa (App móvil, web o de escritorio).
- Estudiar las distintas alternativas comentadas para la localización de objetos.

ENTREGABLES

Los entregables generados durante el trabajo serán los siguientes:

- Programa de localización.
- Programa para aumentar el conjunto de datos partiendo de imágenes ya anotadas, que devolverá a su vez imágenes anotadas.
- Memoria.
- Análisis de herramientas de anotación y una recomendación de la herramienta a utilizar.
- Análisis de modelos para la localización y una recomendación del modelo a utilizar.
- Documentación.

DISEÑO

En la fase de diseño del proyecto tomaremos todas las decisiones necesarias para detallar cada una de las etapas del proceso de implementación del modelo, es decir, explicaremos cómo vamos a llevar a cabo el proyecto. En esta sección se decide cómo abordaremos el problema, cómo obtendremos las imágenes necesarias y cómo las anotaremos, cómo aumentaremos el conjunto de imágenes inicial, qué haremos con esos datos, cómo construiremos el modelo y, finalmente, cómo lo evaluaremos.

CÓMO ABORDAR EL PROBLEMA

El problema de la localización se sitúa en el ámbito del aprendizaje automático. ¿Pero, qué es el aprendizaje automático? ¿Cómo funciona? Esta sección la dedicamos a entender bien el problema al que nos enfrentamos, y el proceso que deberemos seguir para superarlo.

El aprendizaje automático [19] (en inglés, *Machine Learning*) es la rama de la inteligencia artificial dedicada a desarrollar técnicas que permitan a las computadoras *aprender* por sí solas [20]. Es decir, consiste en crear programas capaces de generalizar comportamientos a partir de un conjunto de datos, tal y como hacemos los seres humanos.

Pero el Aprendizaje Automático no siempre funciona de la misma manera, sino que hay varios algoritmos diferentes agrupados en categorías en función de la salida que producen [21]. Algunas de ellas son: aprendizaje supervisado, aprendizaje no supervisado, aprendizaje semi-supervisado, aprendizaje por refuerzo, transducción y aprendizaje multitarea (para más información acerca de los tipos de aprendizaje ver el Apéndice I). Para este proyecto en concreto utilizaremos el aprendizaje supervisado.

- **Aprendizaje supervisado**

En los problemas de aprendizaje supervisado, el algoritmo se entrena a partir de datos etiquetados correctamente (en nuestro caso, imágenes anotadas con la posición del violín indicada). En este tipo de problemas, cuanto mayor sea el conjunto de entrenamiento mejor se ajustará el modelo. Una vez entrenado se le pasan nuevos datos al modelo, pero esta vez sin etiquetar, y será este quien etiquete dichos datos, valiéndose de la experiencia adquirida durante el proceso de entrenamiento. La imagen 7 muestra las fases que se llevarán a cabo durante el proyecto hasta poder obtener la localización de un violín en una imagen.

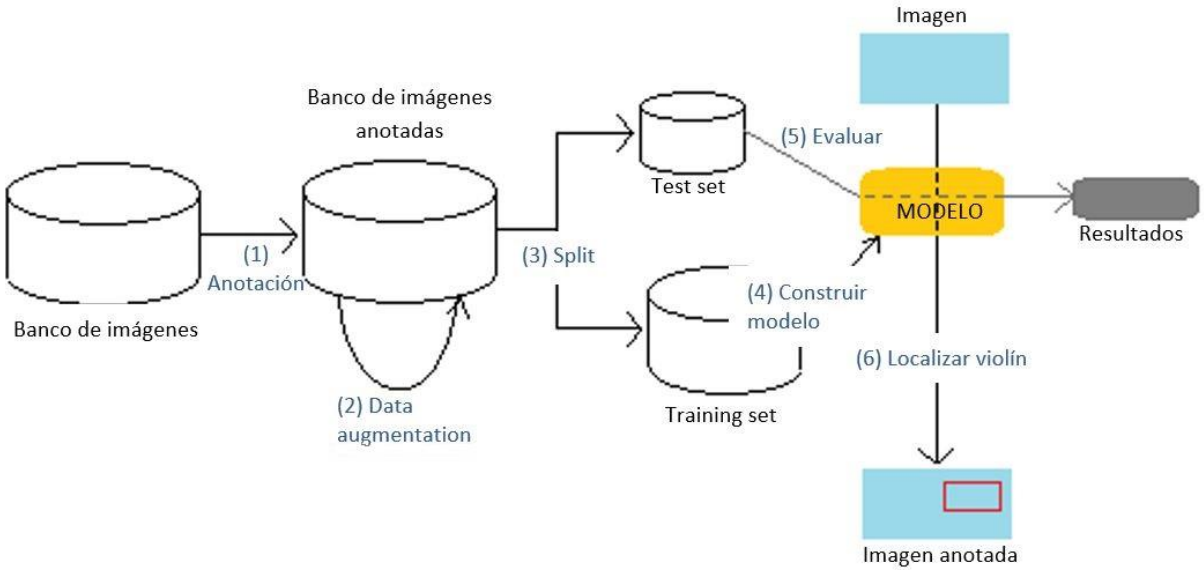


Imagen 7: Estructura de la fase de implementación del proyecto.

OBTENCIÓN DE IMÁGENES

Se ha decidido que el tamaño del dataset inicial será de, en total, 200 imágenes. Dado que la precisión no nos preocupa en principio (ya hemos visto que una posible ampliación del trabajo es incrementar dicha precisión hasta un 95%), creemos que 200 es un número adecuado, ya que no es tan pequeño como para dar problemas durante el entrenamiento, ni tan grande como para dárnoslos durante la anotación, y, además, es aproximadamente el tamaño usado en otros proyectos [18].

Sin embargo, para que el tamaño del dastaset sea eficiente en términos de precisión obtenida, las imágenes no pueden escogerse aleatoriamente. Así, para poder cumplir con el segundo requisito funcional, se construirá el dataset con imágenes variadas y que cumplan con las condiciones establecidas en el mismo.

Las imágenes serán obtenidas desde diferentes fuentes, en concreto: google imágenes [22], Pixelabs [23] e Instagram [24].

ANOTACIÓN DE IMÁGENES

Como ya hemos comentado, el resultado de aplicar nuestro programa a una imagen será una imagen anotada con la posición del violín. A lo que durante esta memoria nos referiremos como imagen anotada no es más que un par de objetos que representan la imagen por un lado, y el rectángulo que indica la posición donde se encuentra el violín por otro. Habitualmente, dicho par de objetos quedan plasmados en un archivo xml que utiliza el formato Pascal Voc [25] como el que muestra la imagen 8. Cuando construyamos el modelo veremos si este formato es o no adecuado para las muestras del conjunto de entrenamiento, ya que existen otros formatos para representar una imagen anotada.

Imagen 8: Aspecto del archivo xml generado tras la anotación de la imagen 1.jpg. La imagen contiene un violín y se localiza en el rectángulo cuya esquina superior izquierda es el punto (243,176) y la inferior derecha el (641,301). Además, se anotan también el nombre de la imagen, la carpeta que la contiene, la ruta completa y sus dimensiones (anchura, altura y número de canales).

La cuestión ahora es: ¿cómo vamos a anotar las 200 imágenes que tenemos? Lo primero que se nos ocurre es hacerlo a mano: dada una imagen, encontramos a ojo el punto superior izquierdo y el punto inferior derecho del menor rectángulo (horizontal) que lo encuadra por completo. Una vez obtenido estos puntos, hayamos sus coordenadas (en algún visualizador de imágenes que lo permita, como, por ejemplo, GIMP) y construimos el archivo xml. Pero, esta tarea puede resultar demasiado costosa, sobre todo si tratamos con un gran conjunto de imágenes. Por ello, la manera más habitual de anotar una imagen es utilizar una herramienta manual de anotación de imágenes. Esta sección la dedicamos al análisis de dichas herramientas, con el fin de obtener una recomendación sobre la herramienta más apropiada en nuestro caso. Las tablas 8.1 y 8.2 recogen las principales características de las que se han estudiado después de realizar una búsqueda exhaustiva en Google. Queremos encontrar una aplicación libre, que funcione en Windows o Linux, que sea fácil de instalar y usar, y cuya salida pueda usarse con facilidad. Para realizar la búsqueda hemos utilizado términos como los siguientes: *image annotation tools*, *annotate images*, *programs for image annotation*, o *herramientas de anotación de imágenes*.

Herramienta	Plataformas/ Requisitos	Formato(s) de salida (imagen anotada)	Facilidad de instalación/uso	Utilidad para nuestro problema
Alp's Labeling Tool (ALT) [26]	Windows 10, Ubuntu 14.04 (bajo Fiji).	.txt	Media.	Útil para trabajar con Detectnet[3] o KITTI[4] [27]. Velocidad de etiquetado baja.
Annotorius [28]	JavaScript, HTML, CSS.	Imagen: el mismo formato que la imagen de entrada.	Media.	Poco útil en nuestro caso (hay que incorporarlo en una página web)
Fast Image Data Annotation Tool (FIAT) [29]	C++ (requiere OpenCV y Google Protobuf).	CSV en formato RotatedRect (se anotan el punto central del rectángulo y la orientación del mismo)	Media.	No recomendada. Desconocemos cómo convertir los CSV en formato RotatedRect a formato .record (formato para Tensorflow).
LabelImg [30]	Linux, Windows.	.xml	Fácil.	Recomendada. Velocidad de anotación media-alta y facilidad alta.
LabelMe [31]	Linux, Windows. Requiere un servidor Apache.	.xml.	Media-alta. Requiere un servidor web.	Fácil de utilizar y velocidad de anotación media-alta.
LEAR [32]	Linux (en teoría también para Windows pero no se asegura).	.xml	Media (requiere instalar otros paquetes).	Fácil de utilizar y velocidad de anotación media. Instalación algo problemática.
RectLabel [33]	Mac OS X.	----	----	No nos sirve porque está pensada para Mac.

Tabla 8.2: Características de algunas de las herramientas de anotación de imágenes.

[3] Detectnet es una red neuronal profunda para la detección de objetos en DIGITS (*Deep Learning GPU Training System*), la aplicación web para la formación de modelos de aprendizaje profundo.
[4] KITTI es una base de datos utilizada en la investigación en robótica móvil y conducción autónoma.

Herramienta	Plataformas/ Requisitos	Formato(s) de salida (imagen anotada)	Facilidad de instalación/uso	Utilidad para nuestro problema
VGG [34]	Ubuntu: uso online (Chrome). Windows: uso offline (archivo comprimido).	CSV y JSON	Fácil.	La anotación online es sencilla, pero el formato de exportación csv no se puede modificar para adaptarlo a nuestro problema. Las etiquetas son distintas.
JSoda [35]	Aplicación Javascript..	KITTI (descargable en texto).	Media.	El formato de exportación no nos sirve. Está pensada para trabajar con DIGITS. Es poco utilizada.
Philosys Label Editor [36]	Windows 7 y 10.	---	---	Aplicación potente pensada para trabajar con vídeos. Más de lo que necesitamos.
Simple Image Annotator [37]	Linux.	CSV.	Fácil (se clona el repositorio y se trabaja en el navegador).	Es muy fácil de utilizar, aunque los rectángulos de localización son difíciles de ajustar. El archivo .csv de exportación no se ajusta exactamente a nuestro problema pero el programa es muy sencillo y permite ajustar las etiquetas.

Tabla 8.2: Características de algunas de las herramientas de anotación de imágenes.

Tras haber analizado las principales características de las herramientas de anotación escogemos **LabelImg** como la más adecuada en nuestro caso. La facilidad de instalación y la velocidad de anotación son el motivo principal de nuestra elección. También Simple Image Annotator podría servir, pero la anotación es un poco más imprecisa, lo que hace que nos decantemos por la otra herramienta.

AUMENTO DE DATOS

Antes de pasar a explicar los modelos de aprendizaje, es necesario hablar de un concepto llamado *data augmentation,* que permite aumentar el conjunto de datos con el objetivo de mejorar la calidad de los modelos.

La clave para la obtención de un buen modelo de aprendizaje automático es tener abundante información (en nuestro caso un gran conjunto de imágenes) y de alta calidad. Pero pocas veces disponemos de tanta información, y conseguirla puede ser una tarea muy costosa. Una forma de evitar la falta de datos es aumentar de manera artificial el conjunto que ya tenemos. Aquí es donde entran en juego las técnicas de data augmentation, que pueden multiplicar el tamaño del conjunto

de entrenamiento más de 10 veces [38]. Es más, puede que el modelo obtenido sea incluso más robusto (evitando el sobreajuste[5]) [38].

Hay muchos enfoques para aumentar los datos. Los más simples incluyen la aplicación de transformaciones en los datos existentes (imagen 8): girar la imagen original, voltearla, cambiar las condiciones de iluminación, recortarla, distorsionar los contrastes, convertirla a los espacios de color HSV o LAB, quitarle el color, aplicarle la ecualización adaptativa del histograma (AHE), añadir ruido sal y pimienta, erosionarla, aplicarle el desenfoque Gaussiano, posterizarla, etc. (Ver el Apéndice II para más información sobre las transformaciones mencionadas).

En nuestro caso, las alternativas para aplicar dichas técnicas y ampliar el dataset son las siguientes:

- O bien utilizar técnicas de *data augmentation* con las imágenes originales (no anotadas) y posteriormente anotar todas,
- o bien aplicar *data augmentation* con las imágenes ya anotadas.

Imagen 9: Una fotografía de un violín: arriba a la izquierda en su posición original; arriba a la derecha girada 90 grados; abajo a la izquierda volteada horizontalmente; abajo a la derecha saturada.

El problema de la segunda opción es que, en caso de no tener cuidado, la localización sería incorrecta dependiendo del tipo de transformación y habría que modificar la anotación (por ejemplo, si giramos la imagen de un violín, la posición de este cambia, como podemos ver en la imagen 9). Para ello tenemos, a su vez, distintas opciones:

- modificar las anotaciones a mano para cada imagen,

[5] El sobreajuste se produce cuando un modelo está entrenado demasiado "bien", reconociendo los detalles, a menudo insignificantes, de las imágenes del conjunto de entrenamiento, de tal manera que no se comporta bien ante imágenes distintas. Ocurre cuando el conjunto de entrenamiento está formado por imágenes muy parecidas entre sí.

- o bien crear un pequeño programa que aplique data augmentation sobre imágenes anotadas, pero que a su vez anote correctamente las imágenes generadas, dependiendo del tipo de transformación que se aplique.

En este proyecto se ha escogido la última opción, por ser la más rápida para datasets grandes y porque puede reaprovecharse en otros proyectos.

En nuestro caso vamos a aplicar, en principio, las siguientes transformaciones sobre el conjunto de entrenamiento: **voltear** la imagen, añadirle **ruido sal y pimienta** y **quitarle el color** (de las cuales voltearla es la única transformación que modificará la posición del violín en la imagen). Con estas tres transformaciones, ampliaremos el conjunto de datos de entrenamiento de 200 a 800 imágenes. Si, una vez entrenado el modelo los resultados muestran un sobreajuste sobre los datos de entrenamiento, se utilizarán menos imágenes transformadas.

Ya hemos comentado que, dependiendo de la transformación que apliquemos a una imagen, la posición del violín varía en la imagen transformada, y que, para solucionarlo, el mismo programa que aplicará la transformación se encargará de general el xml adecuado. La idea será construir una función genérica que, dada una imagen, una transformación y la posición de un rectángulo, devuelva la posición del rectángulo después de aplicar la transformación a la imagen.

DATASET SPLIT

Para saber si un modelo generaliza de manera correcta y poder dar una valoración de su rendimiento, es necesario probarlo con imágenes que no hayan sido utilizadas para entrenarlo. Para ello, necesitamos más imágenes que cumplan los requisitos detallados, y ya hemos comentado que conseguir dichas imágenes conlleva un coste. Así, una vez tenemos construido nuestro banco de imágenes, la idea es separar, es decir, dejar a un lado, una cantidad suficiente para realizar pruebas posteriores con ellas. Estas imágenes no se utilizarán para entrenar el modelo, ya que, si lo hiciéramos, obviamente, el modelo no fallaría.

En nuestro caso, utilizaremos el **75%** de las imágenes para entrenar al modelo y el **25%** para probarlo. La elección se debe simplemente a que es la separación habitual en estos casos. Otras proporciones comunes son: 90% y 10%, o 80% y 20%.

CONSTRUCCIÓN DEL MODELO

Para poder abordar el problema, uno de los aspectos más esenciales es la elección del modelo que vamos a construir. Como ya hemos comentado, tenemos varias alternativas que analizaremos en esta sección para entender la decisión tomada.

- **Tensorflow**

TensorFlow ofrece diferentes modelos para abordar el problema de la localización. Las alternativas que ofrece son: o bien entrenar un modelo desde cero, entrenándolo con nuestras imágenes; o bien utilizar uno de los modelos preentrenados con el conjunto de imágenes COCO [39], KITTI [40] y Open Images [41]. En el segundo caso, el ajuste de los parámetros del modelo es más rápido, dado

que partimos desde un punto en el que el modelo está optimizado para el caso general y solo tendremos que ajustarlo a nuestra clase concreta. Dado que COCO es el mayor conjunto de datos, deducimos que es el que mejor se ajustará al problema. Así, utilizaremos uno de los modelos preentrenados que ofrece Tensorflow para este conjunto de datos. Éstos se agrupan en dos categoriás: los basados en estructuras **faster R-CNN** [42] y los basados en estructuras **SSD** [43]. Las faster R-CNN se surgieron por primera vez en 2015 y constan de dos redes: una de ellas propone las regiones de interés y la otra usa estas propuestas para detectar objetos. Las SSD están formadas por una sola red que realiza las tareas de localización y clasificación de objetos en un único paso hacia delante, y se presentaron por primera vez en 2016.

Ambos tipos de modelos podrían servirnos, pero los primeros, aunque cuesta más entrenarlos, producen mejores resultados, por lo que nos decantamos por usar como primera opción uno de ellos. Sin embargo, en caso de que estos modelos fallen probaremos con los SSD.

Aunque hemos elegido la librería de Tensorflow, otras librerías como Keras [44] o FAIR Detectron [45] también proporcionan este tipo de algoritmos.

- **Entrenar un modelo de clasificación para la localización**

La siguiente alternativa consiste en utilizar una red aplicada inicialmente para un problema de clasificación y modificada para que, en lugar de predecir una categoría, aprenda los dos puntos (4 coordenadas) que representan la posición del rectángulo; es decir, la salida de la red no será una, sino cuatro, que serán las coordenadas de los dos vértices opuestos del rectángulo. Este tipo de técnica se conoce como *Transfer Learning* [46].

Esta alternativa, como la anterior, conlleva el entrenamiento de una red.

- **Mapas de calor**

La siguiente opción se nos presenta a partir de un artículo localizado en internet sobre capas de agrupación para la localización de objetos [47]. El artículo plantea utilizar una red de clasificación para poder detectar la región en la que la red se fija a la hora de decidir de qué clase es un objeto. Todo ello a partir del mapa de calor que genera la red. La imagen 10 muestra varios ejemplos de los mapas de calor de distintas imágenes de perros.

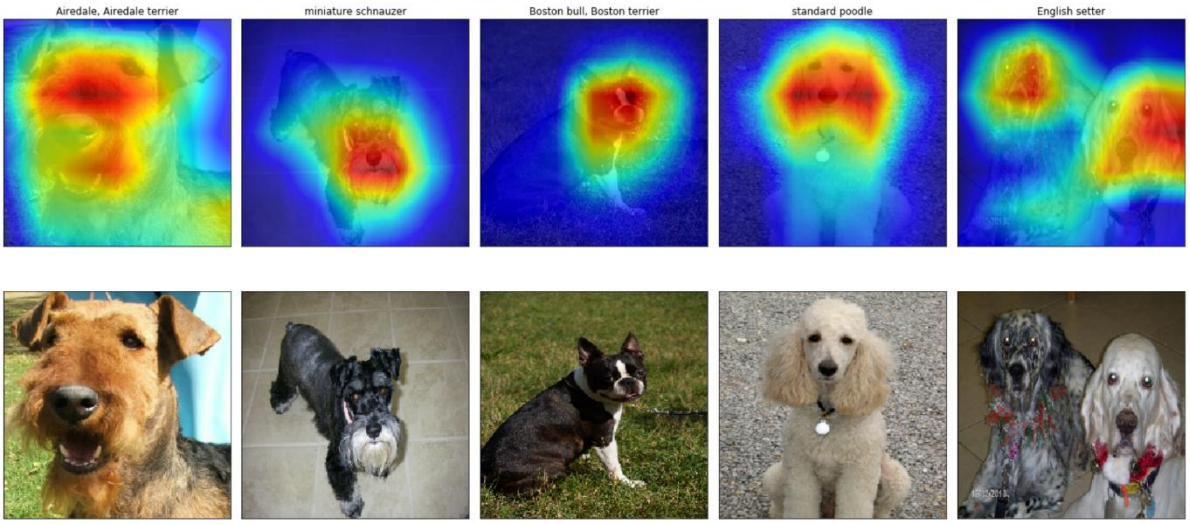


Imagen 10: Ejemplos de mapas de calor de distintas imágenes de perros.

Un mapa de calor (en inglés, heatmap) es una representación gráfica de datos donde los valores individuales contenidos en una matriz (en nuestro caso, la imagen) se representan como colores. Se utilizan para identificar cuáles son los puntos o regiones de interés (en nuestro caso, las zonas de interés para el modelo). Utilizan como forma de representación una termografía, estableciendo una jerarquía de dos polos: emplea colores cálidos (rojo, naranja y amarillo) para mostrar zonas de interés, y una gama de colores fríos (azul y verde) para las zonas de poco interés.

El problema de esta alternativa es que no podremos cumplir con algunos de los requisitos planteados inicialmente para el proyecto. El motivo es que la red de clasificación se fija en el objeto principal de la imagen, y es dicho objeto el que localizaremos. Así, no podremos asegurar el funcionamiento correcto del programa para imágenes en las que el violín no sea el objeto principal, esté mal enfocado, o haya otros objetos en primer plano. Además, se debe cumplir que la red haya sido entrenada para ser capaz de localizar objetos como violines.

- **Segmentación semántica**

La última de las opciones es la de abordar el problema desde el punto de vista de la segmentación semántica. El problema de esta alternativa es que, aunque seguramente sea eficaz, aborda un problema más amplio que el de la simple localización de un objeto en una imagen: no sólo localizamos un objeto, sino todos los que contiene la imagen, y, además, los clasificaremos. Sin embargo, si las demás alternativas no funcionaran, puede ofrecernos una posible solución a nuestro problema clasificando los píxeles de la imagen en dos categorías: violín y *background* (o fondo).

EVALUACIÓN DEL MODELO

Hasta ahora hemos mencionado en varias ocasiones que el modelo se evaluará; que se obtendrán resultados para detectar el modelo que mejor se ajusta al problema; que no imponemos en principio una precisión mínima. Pero, ¿cómo vamos a medir o evaluar el modelo?, ¿qué criterios vamos a seguir? En este punto, debemos establecer una medida para determinar la calidad del modelo en términos de precisión.

Hay varias medidas posibles para este tipo de problemas, pero la que usaremos en este caso es la llamada *intersection over unión* (IOU) [48], es decir, intersección sobre unión. Gráficamente, se trata de lo siguiente: tenemos una foto como la mostrada en la imagen 11, y determinamos que la localización correcta del violín es la que determina el rectángulo rojo. Si suponemos que el modelo da una localización distinta, marcada en la imagen con un rectángulo azul, la medida que proponemos diría que la precisión es el área de la intersección de ambos rectángulos dividida entre el área de su unión. Es decir:

$$\textbf{IOU} = \frac{\text{área intersección entre predicción y localización real}}{\text{área unión predicción y localización real}}$$

Un detalle a tener en cuenta es que, según esta medida, si dos modelos distintos localizaran un violín en una imagen como en las imágenes 12 y 13, el primer modelo sería más preciso, mientras que el segundo tendría una precisión igual a cero. Sin embargo, como podemos ver, el segundo modelo ha localizado perfectamente un objeto muy parecido a un violín.

Imagen 11: Foto de un grupo de música tocando en la calle.

Imágenes 12 y 13: localización de un violín en una imagen de dos modelos hipotéticos diferentes. En rojo la localización real y en azul la predicha por el modelo correspondiente.

Además de la medida que vamos a utilizar, existen muchas otras, algunas de ellas válidas para nuestro problema, y otras no tanto. Por ejemplo, podemos dar por bueno un resultado en el que la mitad de los píxeles (o la proporción que escojamos) predichos están contenidos en la localización real del violín. De esta manera la precisión de un resultado valdría 1 o 0, y la del modelo se extraería haciendo la media de todas ellas.

Sin embargo, hay ciertas medidas que no nos sirven en nuestro caso: si damos por bueno un resultado que contiene la localización real del violín, podríamos estar dando por buena una localización que contuviera toda la imagen, lo cual no tendría sentido. Es por esto que se utiliza la métrica IOU, ya que no solo nos dice si el violín está contenido en la predicción, sino que además nos dice si la predicción es ajustada.

IMPLEMENTACIÓN

Esta sección la dedicamos a detallar el proceso de implementación que hemos seguido a lo largo del proyecto: nombraremos las tecnologías utilizadas; explicaremos el proceso de construcción del dataset, de la anotación de imágenes y del aumento de dicho conjunto de datos que hemos seguido, así como de la creación del modelo de localización, en la explicamos los imprevistos que han surgido y las medidas que hemos tomado ante los mismos.

TECNOLOGÍAS

Para llevar a cabo el proyecto, utilizaremos distintas tecnologías y herramientas, algunas de ellas comunes para las diferentes alternativas expuestas en la fase de diseño, y otras específicas para cada una de ellas:

- Comunes:
  - Python como lenguaje de desarrollo, por ser el lenguaje más utilizado y en el que más librerías están disponibles en el mundo de la visión por computador y el Deep Learning.
  - Pycharm Community como IDE de desarrollo para Python.
  - LabelImg como herramienta para la anotación de imágenes.
  - OpenCV para tratamiento de imágenes.
  - Matplotlib también para tratamiento de imágenes.
  - Ubuntu 16.04 en local como entorno de trabajo.
- Específicas:
  - TensorFlow como librería de algoritmos de visión por computador, aprendizaje automático y localización de objetos.
  - Keras para la segmentación semántica y los mapas de calor.

Es importante comentar las características del equipo que vamos a utilizar para la realización del proyecto, ya que este tipo de problemas requieren de muchos recursos, y de ellos dependen el desarrollo y ejecución de los modelos de Deep Learning:

- Procesador Intel® Core™ i5-4210M CPU @ 2.60GHz (2 cores).
- RAM instalado: 8,00 GB
- Sistema operativo de 64 bits, procesador x64.

CONSTRUCCIÓN DEL DATASET

El conjunto de datos se ha obtenido según lo previsto en el diseño del proyecto. La obtención de las imágenes ha sido más o menos sencilla, aunque el hecho de tener que encontrar violines en diferentes ámbitos o situaciones no ha sido del todo intuitivo. Hemos realizado numerosas búsquedas, sobre todo en Google, de expresiones como las siguientes: *violin*, *violin nature*, *Lindsey*

Strilling, *violin and cello*, *violin jazz band*, etc. Así, hemos conseguido obtener 200 imágenes de violines variadas, cumpliendo con los requisitos necesarios para conseguir entrenar el modelo lo mejor posible.

ANOTACIÓN DE IMÁGENES

Ya hemos comentado en la sección de anotación de imágenes del diseño del proyecto que hay varias herramientas para realizar la anotación de imágenes, pero que la mejor en nuestro caso (y según los criterios elegidos para abordar el problema) es LabelImg, por la sencillez de su instalación y uso, y la rapidez en la anotación de las imágenes. Con esta herramienta hemos anotado las 200 imágenes del conjunto de imágenes inicial en, aproximadamente, 1 hora (hemos querido ajustar los rectángulos con la mayor precisión posible y en algunas imágenes el violín era difícil de acotar). Como ya hemos comentado, las imágenes anotadas se han guardado en formato Pascal Voc xml.

AUMENTO DE DATOS

Como ya hemos comentado en el diseño, el aumento de datos lo haremos a partir de imágenes anotadas (archivos xml). Para ello, hemos creado un pequeño programa (ver archivo Aumentar_dataset.py del Apéndice III:Código) que itera sobre los archivos xml de cierta carpeta de origen, aplica una serie de transformaciones a cada una de las imágenes (las que nosotros queramos) y guarda, no solo las imágenes generadas, sino también el xml generado para cada una de ellas. Las funciones que transforman una imagen en otra se pueden ver en el archivo Funciones_transformar.py del Apéndice III: Código. En nuestro caso hemos implementado más transformaciones de las que hemos utilizado: voltear, quitar el color, salpimentar y girar 90 grados (no utilizaremos esta última).

CREACIÓN DEL PROGRAMA DE LOCALIZACIÓN USANDO TENSORFLOW

En este apartado detallaremos el proceso de creación del programa de localización, incluídos los problemas, impedimentos e imprevistos que han ido surgiendo. Como hemos indicado en el diseño, en un primer momento usaremos uno de los modelos Faster R-CNN que ofrece Tensorflow. Elegimos *faster_rcnn_inception_resnet_v2_atrous_lowproposals_oid* como primera opción.

- **Construcción del modelo faster_rcnn_inception_resnet_v2_atrous_lowproposals_oid**

Empezamos la implementación del proyecto generando un proyecto en Pycharm y construyendo la estructura de directorios necesaria.

Los modelos preentrenados de Tensorflow[6] se descargan en formato tar.gz y contienen: el grafo[7], un checkpoint[8] (formado por tres archivos: model.ckpt.data-00000-of-00001, model.ckpt.index y model.ckpt.meta) y un archivo de configuración (pipeline.config) que ha sido utilizado para generar

[6] Los modelos preentrenados se pueden encontrar en https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md.
[7] El grafo es el modelo preparado para ser utilizado.
[8] Un checkpoint es una estructura que contiene el valor de todas o algunas de las variables de un grafo en un determinado momento. Es utilizado para entrenar un modelo en distintas etapas, cargando los pesos de la última de ellas.

el grafo. A partir de este directorio podremos entrenar el modelo con nuevos datos, ajustando los últimos pesos (los del checkpoint) a nuestro problema.

El primer problema surge cuando el modelo escogido no nos permite, por un error que no conseguimos solucionar, ser entrenado. El error es el siguiente (la imagen no contiene el error completo porque era muy extenso, sino la última parte del mismo):

```
WARNING:root:Variable [InceptionResnetV2/Repeat_2/block8_9/Conv2d_1x1/weights] not availabe in checkpoint
Traceback (most recent call last):
  File "train.py", line 158, in <module>
    tf.app.run()
  File "/usr/local/lib/python2.7/dist-packages/tensorflow/python/platform/app.py", line 48, in run
    _sys.exit(main(_sys.argv[:1] + flags_passthrough))
  File "train.py", line 154, in main
    worker_job_name, is_chief, FLAGS.train_dir)
  File "/usr/local/lib/python2.7/dist-packages/tensorflow/models/research/object_detection/trainer.py", line 255, in train
    init_saver = tf.train.Saver(available_var_map)
  File "/usr/local/lib/python2.7/dist-packages/tensorflow/python/training/saver.py", line 1218, in __init__
    self.build()
  File "/usr/local/lib/python2.7/dist-packages/tensorflow/python/training/saver.py", line 1227, in build
    self._build(self._filename, build_save=True, build_restore=True)
  File "/usr/local/lib/python2.7/dist-packages/tensorflow/python/training/saver.py", line 1251, in _build
    raise ValueError("No variables to save")
ValueError: No variables to save
```

Lo que sucede, aparentemente, es que el modelo no encuentra muchas de las variables que deberían estar almacenadas en el checkpoint a partir del cual es entrenado. Por tanto, no podemos solucionar el problema ya que parece ser que el checkpoint que nos proporcionan es incompleto. Así que probamos a utilizar uno de los modelos preentrenados en el conjunto de imágenes COCO, que, aunque no contiene al violín entre sus clases, es el modelo que utiliza el tutorial que seguimos [18] y que parece que funciona aunque el objeto no esté entre las clases de COCO. El modelo escogido es: ssd_mobilenet_v1_coco, es decir, se basa en el algoritmo SSD.

- **Construcción del modelo ssd_mobilenet_v1_coco**

Antes de nada, vamos a detallar los pasos que hay que seguir para entrenar un modelo preentrenado de tensorflow a partir del tar.gz descargado y de nuestras imágenes anotadas:

- Modificar el archivo .**config**: el fichero ssd_mobilenet_v1_coco.config determina la configuración de los parámetros que va a utilizar el modelo. Para poder reentrenar el modelo debemos ajustar los siguientes parámetros:

    - fine_tune_checkpoint: ruta completa al modelo que hemos descargado como checkpoint.
    - Dentro de train_input_reader: input_path: ruta completa al fichero train.record que hemos generado a partir de las imágenes anotadas. Dicho fichero, junto con el de test.record, lo generamos con un pequeño programa que convierte la información de los archivos xml de las imágenes anotadas en un archivo CSV, y posteriormete en un archivo .record, que es el formato que utilizan los modelos de Tensorflow para leer datos.

- Dentro de train_input_reader: label_map_path: ruta completa al archivo .pbtxt de mapeo de clases, necesario porque el modelo trabaja con índices de clases y no con nombres.
- Como nosotros solo nos vamos a preocupar de la clase *violín*, es la única que indicaremos en el archivo.
- Dentro de eval_input_reader: input_path: ruta copleta al archivo text.record.
- Dentro de eval_input_reader: label_map_path: ruta completa al archivo .pbtxt de mapeo de clases.

– Comando para entrenar el modelo:

```
python RUTA_COMPLETA_A_TENSORFLOW/models/object_detection/train.py --
logdir=RUTA_CARPETA_LOG --
pipeline_config_path=RUTA_COMPLETA_MODELO/ssd_mobilenet_v1_coco.config --
train_dir=RUTA_COMPLETA_CARPETA_ENTRENAMIENTO
```

Cuando empezamos a entrenar el modelo, cada paso de ajuste de pesos tarda cerca de un cuarto de hora: al cabo de unas 2 horas de entrenamiento hemos conseguido llegar al paso 9 y la pérdida sigue siendo de en torno a 6. La pérdida es la medida que se utiliza para saber cómo de bien está entrenado el modelo en cada momento, y se calcula sumando los cuadrados de las diferencias entre las precisiones y el valor real. Para considerar que el modelo está bien entrenado, la pérdida tiene que ser lo más cercana a 0 que podamos (aunque rara vez suelen bajar de 1), por lo que una pérdida de 6 indica que el modelo no está ajustado aún a nuestro problema. Además, después de cierto tiempo (en torno a unas 2 horas), el ordenador no puede más y se queda colgado (le estamos pidiendo demasiado) (ver los requisitos del proyecto en la sección de Análisis). Esto ocurre porque, aunque el ordenador que utilizamos tiene unas características bastante buenas, no está preparado para ofrecer la potencia que este tipo de problemas requiere.

El motivo por el cual un paso tarda tanto en ejecutarse es que en el fichero de configuración el tamaño del batch[9] es de 25. Para poder entrenar el modelo durante más tiempo sin interrupciones y ver además el progreso del entrenamiento en intervalos más pequeños de tiempo, reducimos el tamaño del batch a 5. Así, conseguimos poner a entrenar el modelo de tal manera que cada uno de los pasos tarda entre 3 y 4 segundos.

Sin embargo, la velocidad de entrenamiento sigue siendo baja: al no tener una GPU, hemos tenido que dejar al modelo entrenándose durante 4 noches enteras (unas 32 horas) para conseguir un punto (checkpoint) con una pérdida cercana a 1. Una vez tenemos entrenado el modelo, los resultados obtenidos no son los esperados: hemos probado a predecir la localización de varias imágenes del conjunto de prueba y los resultados mostrados parecen aleatorios y para nada los correctos.

[9] El tamaño de batch (batch_size) en el entrenamiento de un modelo indica el tamaño del subconjunto de datos que se usan cuando se realiza un descenso de gradiente. Es decir, si, por ejemplo, lo ajustamos a 100, el modelo se va entrenando primero con las 100 primeras imágenes, después con las siguientes 100, y así sucesivamente. De esta manera, el entrenamiento requiere de menos memoria, y, cuanto menor sea el tamaño del batch, más rápido se entrenará el modelo. El mayor inconveniente es que tarda más en llegar al mejor resultado, ya que el gradiente varía más en cada iteración (ya que cambiamos los datos).

Imágenes 14 a 17: Resultados obtenidos a partir del modelo para imágenes del conjunto de entrenamiento.

Probamos a pasarle algunas de las imágenes del conjunto de entrenamiento al modelo. En teoría, el modelo ha sido entrenado para aprender la localización de los violines en esas imágenes, por lo que, si la pérdida es cercana a 1, la predicción sobre las mismas debería ser casi perfecta. Sin embargo, los resultados en este caso nos resultan incomprensibles. Sigue sin acercarse al resultado correcto; es más, las localizaciones son totalmente erróneas (ver imágenes 14 a 17).

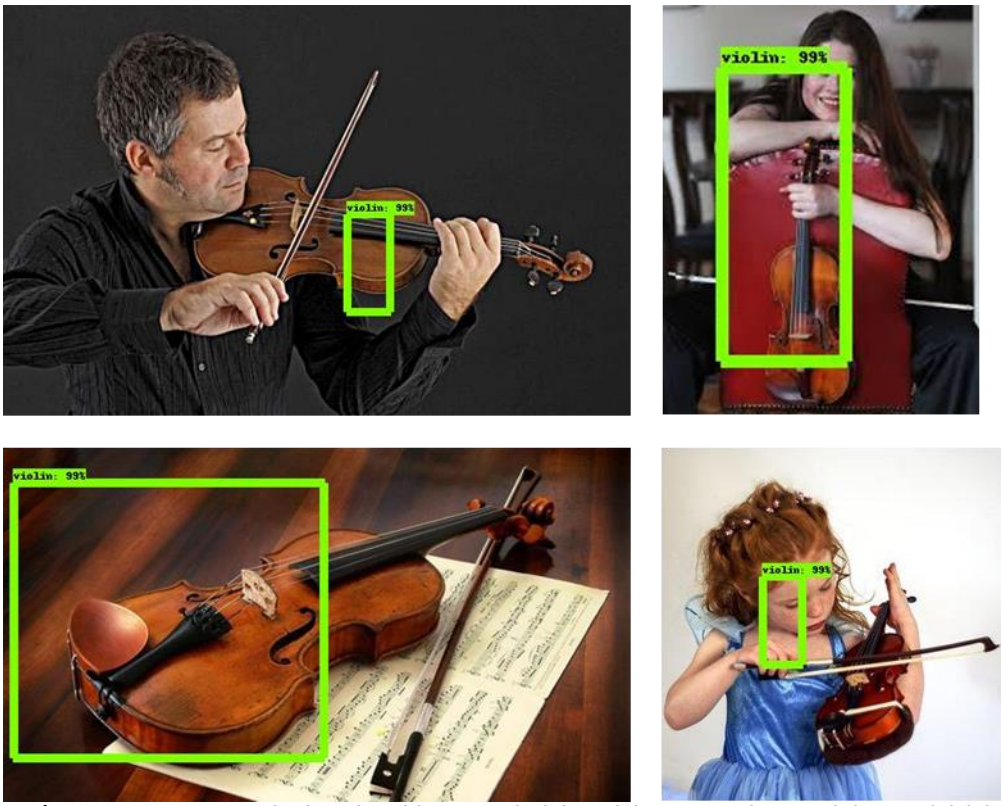
Estos resultados nos llevan a una tercera fase de repaso del proceso seguido hasta ahora: desde la anotación del dataset inicial, hasta las pruebas del modelo, pasando por la fase de aumento de datos.

- **Repaso de los pasos seguidos. Búsqueda del problema surgido**

Esta es la fase más tediosa de la implementación. Hasta ahora hemos ejecutado cada una de las fases con sumo cuidado, realizando pruebas tras cada pequeño avance, para asegurarnos de no pasar por alto ningún error. Sin embargo, nos hemos encontrado con resultados no esperados, lo que nos hace pensar que hemos fallado en algún punto del proceso.

Para asegurarnos de que el problema no ha estado en la fase previa de anotación de imágenes, hemos creado un pequeño programa (función *visualizar_imagen_anotada(xml)* del archivo *funciones_auxiliares.py* del Apéndice III: Código) para visualizar una imagen anotada a partir del archivo xml. Probamos tanto las imágenes del conjunto de datos inicial como las generadas en el proceso de aumento de datos. Los resultados parecen correctos. Sin embargo, realizamos también la comprobación desde los propios archivos xml. Abrimos las imágenes con un visor de imágenes que muestre los píxeles y comprobamos la posición del violín (a ojo) con la indicada en el fichero xml. En principio (no hemos comprobado todas las imágenes, pero dado que el proceso ha sido el mismo para todas, no lo vemos necesario), parece que los datos con los que hemos trabajado son correctos.

Visto que el problema no está, aparentemente, en la anotación de las imágenes, intentamos otra aproximación. Aunque no tenga mucho sentido, es probable que el haber utilizado imágenes creadas a partir del aumento de datos haya dado algún tipo de problema. Por ello, probamos a entrenar otra vez el modelo con las 150 imágenes del conjunto de entrenamiento inicial. Los resultados obtenidos tras conseguir una pérdida cercana a 1 después de varios días han sido mejores. Algunos de los mismos podemos verlos en las imágenes 18 a 21.



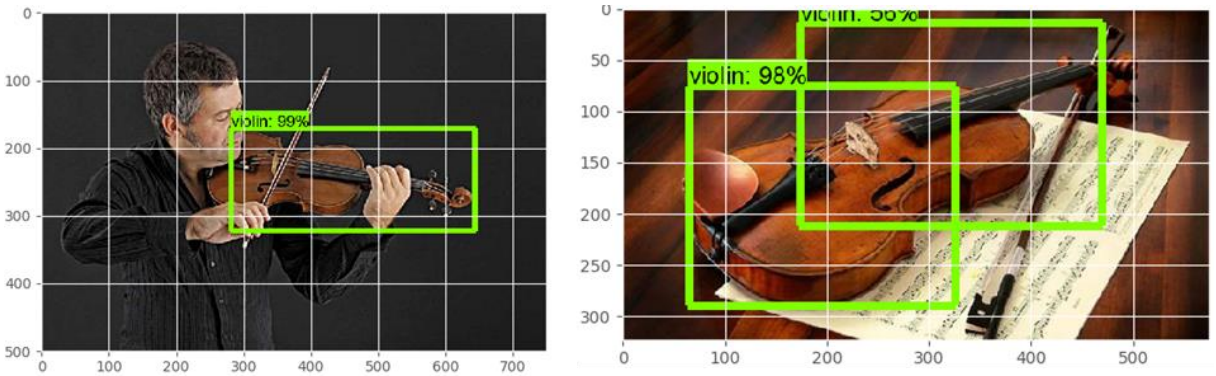
Imágenes 18 a 21: Resultados obtenidos a partir del modelo entrenado con el dataset inicial.

Aunque los resultados obtenidos en este caso son algo más cercanos a la realidad, no podemos decir aún que sean los esperados. Una de las imágenes mencionadas anteriormente (abajo a la derecha) nos muestra que el modelo considera que el violín está en el rectángulo verde con una probabilidad del 99%, lo cual es totalmente erróneo. Además, tenemos que tener en cuenta que le hemos pasado las imágenes con las que el modelo ha sido entrenado, lo cual no debería dar, apenas, lugar al error.

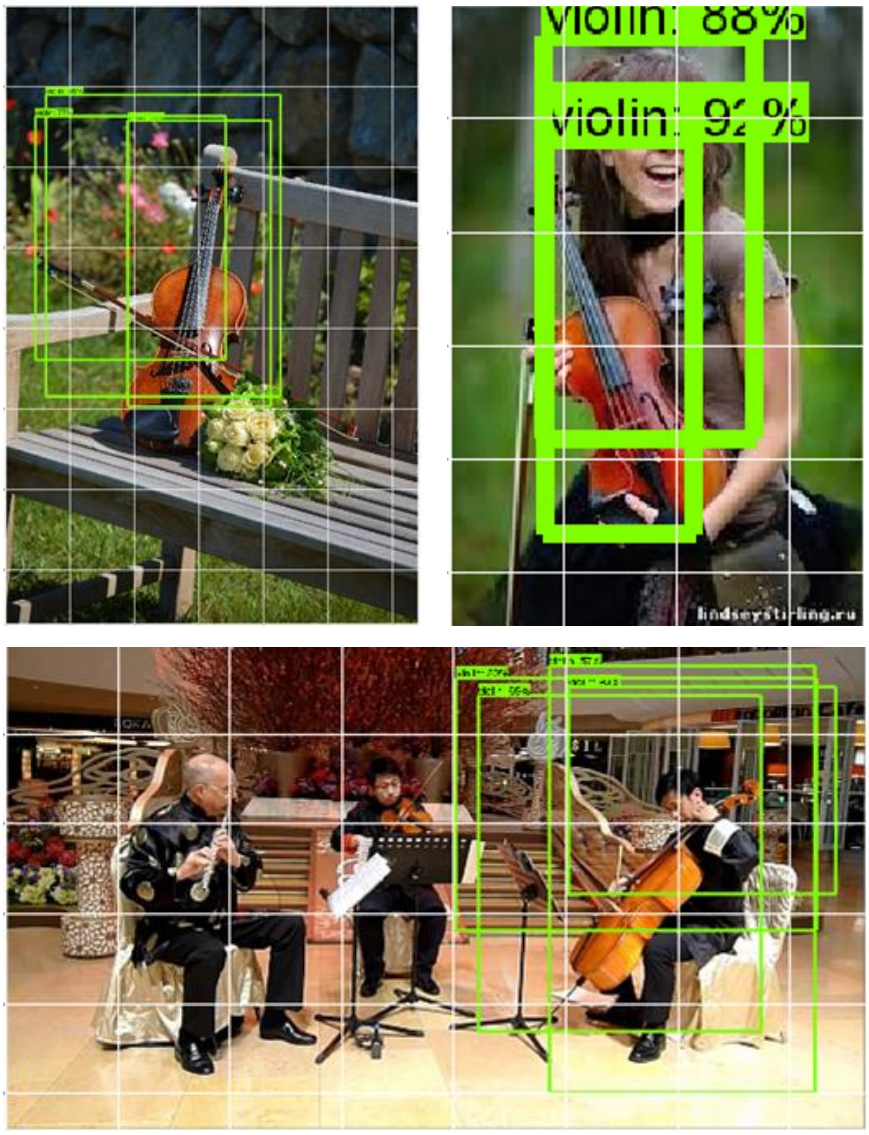
Siguiendo el consejo del tutor del proyecto, hemos probado a entrenar el modelo con tan solo 5 imágenes, etiquetadas de nuevo. De esta forma, el modelo se entrenará mucho más rápido (aunque, obviamente, se sobreajustará a las imágenes del conjunto de entrenamiento), y veremos si el problema persiste. Elegimos 5 imágenes bastante claras (con el violín en primera plana y bien enfocado) y las hemos vuelto a anotar con LabelImg. Entrenamos el modelo hasta conseguir una pérdida cercana a 1. Los resultados son pésimos: el modelo ni siquiera muestra un rectángulo en las imágenes de entrenamiento; es decir, decide que ninguno de los objetos de interés de las imágenes puede ser un violín.

Llegados a este punto, empezamos a pensar que el problema puede deberse a la propia librería. Nos damos cuenta de que utilizando los últimos archivos *export_inference_graph.py* y *train.py* de

Tensorflow los resultados varían: los resultados de dos de las cinco imágenes de entrenamiento los vemos en las imágenes 22 y 23, y los obtenidos para tres imágenes distintas en las imágenes 24 a 26.



Imágenes 22 y 23: resultados obtenidos para dos imágenes del conjunto de entrenamiento.



Imágenes 24 a 26: resultados para tres imágenes no contenidas en el conjunto de entrenamiento.

Los resultados obtenidos con las imágenes del conjunto de entrenamiento se ajustan bastante bien a la realidad, y, aunque los obtenidos con imágenes distintas son algo más imprecisas, también son bastante aceptables. Así, es posible que hayamos encontrado el motivo del error, que no sea otro si no el de utilizar archivos de la librería desactualizados e incorrectos.

Así, probamos a volver a entrenar y probar el modelo con los nuevos archivos de *generate_tfrecord.py* y *train.py* (cambiando simplemente un par de comandos) y con las 200 imágenes del conjunto de entrenamiento inicial (aún no usamos las 600 del dataset aumentado porque el entrenamiento es mucho más costoso, y, de momento, solo queremos saber si hemos encontrado realmente el problema y su solución). Lamentablemente, los resultados no son los esperados: el modelo sigue prediciendo de manera totalmente incorrecta las posiciones de los violines. Los resultados obtenidos se muestran en las imágenes 27 a 31.

Imágenes 27 – 31: Resultados obtenidos para algunas de las imágenes de entrenamiento con el modelo entrenado con los nuevos archivos generate_tfrecord.py y train.py.

Llegados a este punto, y, siguiendo el asesoramiento del tutor, decidimos cambiar de rumbo y dejar de intentar solucionar un problema al que, muy posiblemente, no encontremos solución. Consideramos que no tenemos los conocimientos necesarios sobre la librería como para poder encontrar el problema: es una librería nueva (la lanzaron el 9 de noviembre de 2015) y tiene escasa documentación. Además, aunque hay otras librerías que implementan los mismos algoritmos, estudiarlas puede ser muy costoso y Tensorflow es la "más" fiable, por ser la más utilizada hoy en día. El porcentaje de avance del proyecto que tenemos en este momento es bastante alto (en cuanto a horas), y la utilización de otra librería podría llevarnos a no poder obtener unos resultados satisfactorios en lo que queda de proyecto. Por tanto, a partir de ahora dejaremos a un lado esta vía e intentaremos construir el programa de localización desde alguno de los otros punto de vista sugeridos en la fase de diseño.

LOCALIZACIÓN CON MAPAS DE CALOR

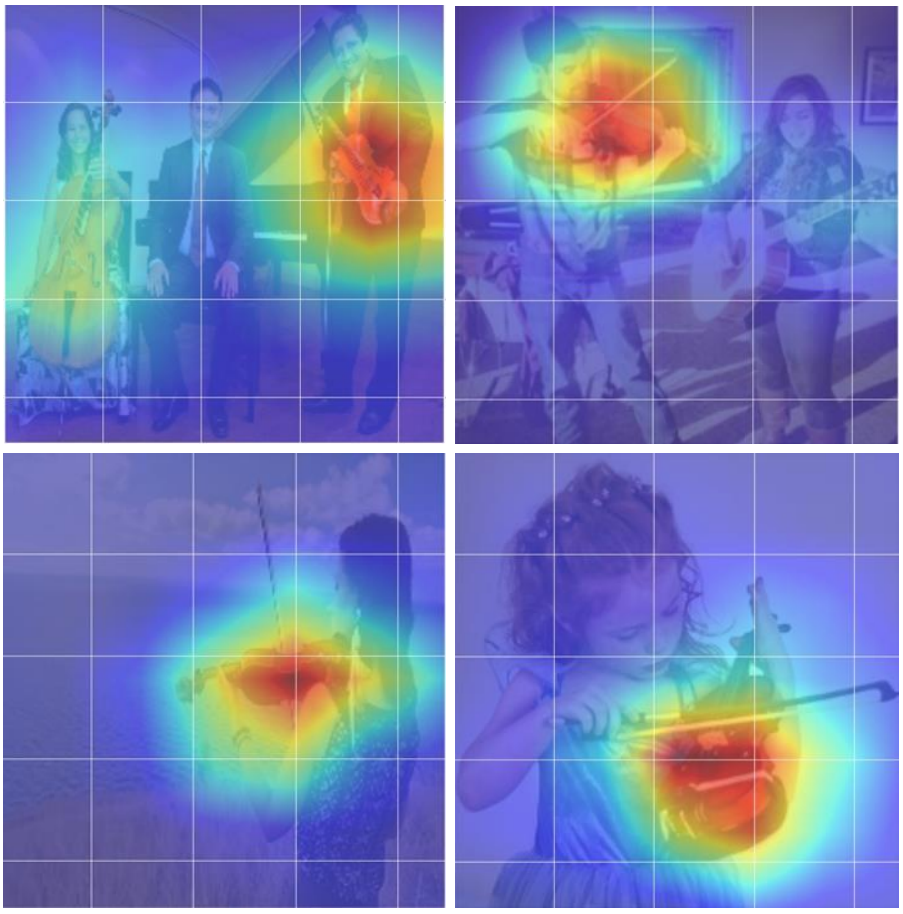
En la fase de diseño hemos comentado que una de las alternativas a las librerías de localización utilizando Deep Learning era la de entrenar un modelo de clasificación para el problema de la localización. Sin embargo, teniendo en cuenta que esta alternativa también conlleva un entrenamiento y, por consiguiente, un consumo de recursos elevado, y que tal y como nos ha pasado con Tensorflow, los resultados pueden no ser los esperados, la descartamos. Así, nos centramos en la siguiente alternativa planteada: la localización a partir de mapas de calor de un modelo de clasificación ya entrenado.

Imágenes 32 a 36: Posibles imágenes de prueba para el modelo construido a partir de mapas de calor.

Ya hemos comentado en la fase de Diseño que el modelo no podrá localizar violines en cualquier tipo de imagen que contenga uno. Las imágenes 32 a 36 muestran algunas de las posibles entradas que daremos al programa para evaluarlo de tal manera que el resultado sea representativo. En todas ellas el violín es el único objeto importante, y, por tanto, el objeto en el que se fijará nuestra red para decidir la clase a la que pertenece.

Hasta ahora solo hemos comentado por encima cómo vamos a hacer para utilizar un clasificador como localizador, pero no cómo vamos a hacerlo. El programa construido para tal fin se puede ver en el archivo *evaluacion.py* del Apéndice III: Código.

Los mapas de calor que se generan a partir del modelo de Google (un modelo ResNet50 implementado en Keras) [49] para las imágenes de nuestro conjunto de datos tendrán el aspecto de las imágenes 37 a 40.



Imágenes 37 a 40: aspecto de los posibles mapas de calor generados por el modelo a partir del cual trabajaremos.

Una vez obtenido el mapa de calor que nos da la red, lo único que tendremos que hacer es localizar la región más caliente (donde se encontrará el objeto de mayor interés) y cubrirla con el menor rectángulo posible. Las áreas de interés las encontraremos usando la técnica de thresholding y el rectángulo lo generaremos haciendo uso de los contornos de OpenCV.

- **Thresholding**

La técnica de thresholding consiste en transformar los píxeles de una imagen a los valores blanco o negro (255 o 0, respectivamente) en función del valor actual del pixel. Lo único que necesitamos es fijar un valor entre 0 y 255 a partir del cual los píxeles que lo superen se volverán blancos y los que no, negros. De esta manera transformamos un mapa de calor en una imagen negra con regiones blancas. Después, nos centraremos en la región más grande como candidata a localizar el violín.

Los valores de thresholding utilizados en nuestro caso han variado entre 10 y 60; utilizando valores fuera de este rango obteníamos o bien imágenes negras, o bien regiones blancas demasiado grandes. En la sección de evaluación se muestran los resultados obtenidos a partir de diferentes valores.

- **Contours**

Para hallar el contorno[10], en nuestro caso rectangular, de una región blanca, OpenCV nos ofrece una serie de funciones fáciles de utilizar.

Una vez que tenemos los contornos rectangulares de nuestras zonas de interés, elegimos la mayor de ellas y ya tenemos nuestro rectángulo.

Aunque durante el proceso no han surgido grandes impedimentos, los resultados obtenidos no han sido del todo satisfactorios (podemos verlos en la sección de Evaluación). En muchos casos, la localización es bastante imprecisa y abarca mucho más que el violín, lo que hace que no podamos dar el modelo como definitivo. Por ello, nos planteamos construir otro modelo a partir de la última de las opciones comentadas: la segmentación semántica.

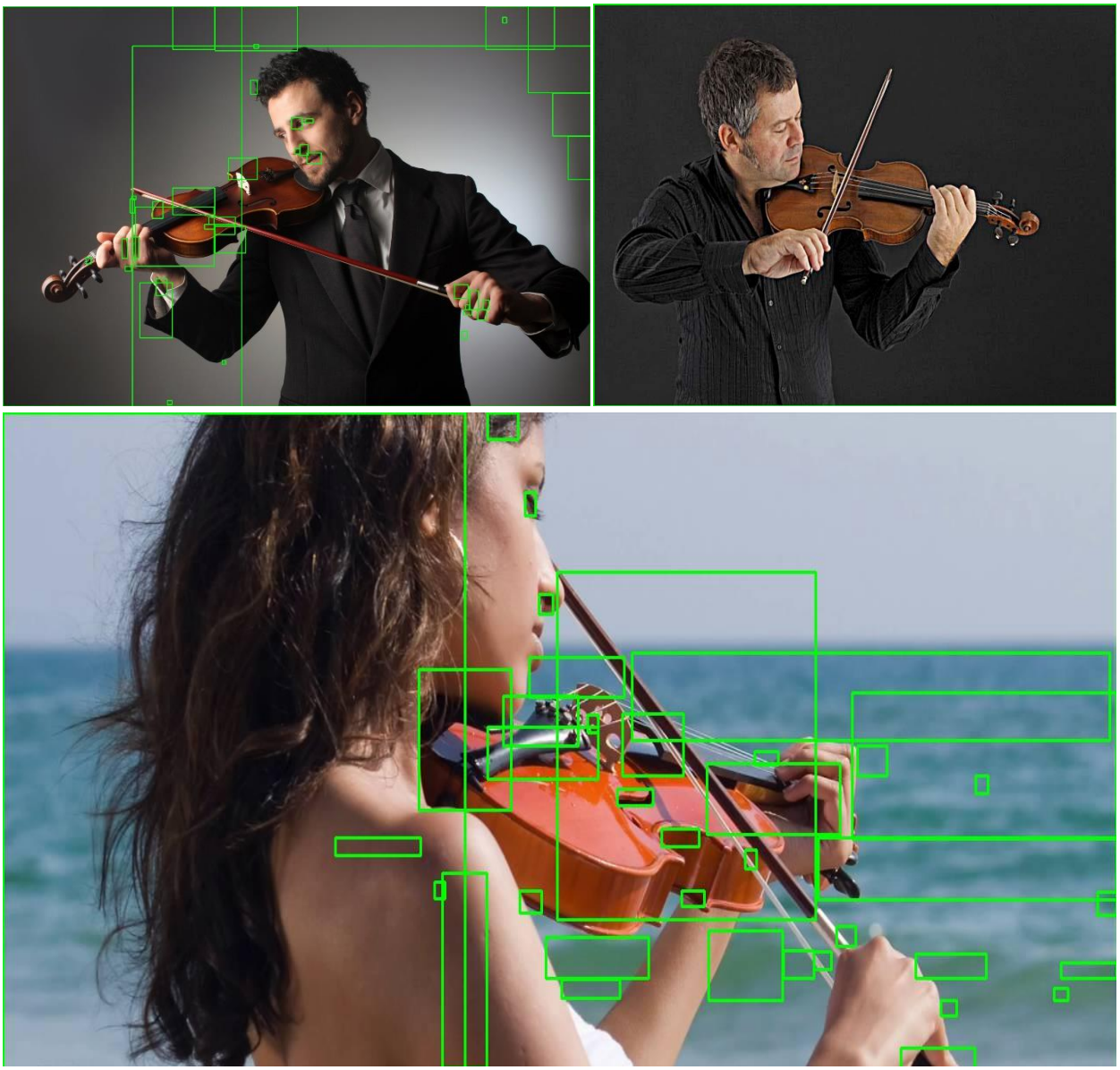
SEGMENTACIÓN SEMÁNTICA

En la sección de Diseño, planteábamos la segmentación semántica como última opción para abordar el problema de la localización. Para ello, hemos seguido un proyecto ya existente y localizable en Github[11].

Aunque esta aproximación también supone entrenar un modelo, esta técnica ya ha sido utilizada de manera exitosa por el grupo de informática para detectar estomas en imágenes de plantas y, por lo tanto, tenemos ciertas garantías de que puede funcionar de manera correcta para el problema que abordamos.

Tras implementar el modelo sin mayor dificultad y entrenar la red obteniendo una pérdida inferior a un 0,3, los resultados obtenidos son peores que los obtenidos con los mapas de calor. Podemos ver algunos de ellos en las imágenes 41 a 43.

[10] El contorno de una región en una imagen es la curva que une todos los puntos continuos que tienen el mismo color o intensidad a lo largo del límite.
[11] El proyecto está localizable en https://github.com/joheras/SemanticSegmentationForObjectDetection.



Imágenes 41 a 43: Resultados obtenidos con la segmentación semántica. En la imagen superior derecha el modelo no ha reconocido ningún objeto. En las otras, reconoce muchos, pero ninguno es el violín completo.

Dado que el proyecto está llegando a su fin y que nuestros conocimientos sobre la segmentación son escasos, no podemos intentar mejorar el modelo por esta vía. El único modo habría sido tratar de quedarnos con uno solo de los objetos que reconocía el modelo (optando, seguramente, por el mayor de ellos), pero los resultados nos muestran que este cambio no nos ofrecería resultados satisfactorios, ya que, al menos en las imágenes del conjunto de entrenamiento, ninguno de los objetos reconocidos coincide con el violín. Además, la red que estamos usando está pensada para detectar objetos pequeños (como células, por ejemplo) dentro de imágenes grandes, y es posible que esta sea la razón de que no se adapte bien a nuestro problema.

EVALUACIÓN DEL MODELO

Esta sección la dedicamos a la evaluación del modelo construido. En la fase de diseño ya hemos introducido la medida (IOU) que utilizaremos para medir la precisión del mismo. Sin embargo, la librería de Tensorflow no nos ofrece el código para medir la precisión en estos términos, por lo que hemos tenido que crear a mano una función para ello. Podemos ver la construcción de dicho programa en el archivo *evaluar_IOU_media.py* del Apéndice III: Código.

La función a la que llamaremos para cada una de las imágenes predichas es la llamada *obtener_IOU*, que tiene como parámetros de entrada la imagen predicha y las coordenadas de los dos puntos que representan el rectángulo predicho. Con ellos calcula la IOU de la predicción correspondiente cogiendo los archivos xml de la imagen anotada a mano y calculando la intersección y la unión del rectángulo indicado en la misma, y del rectángulo predicho. Desde la función de *evaluacion.py* de nombre *predecir_imagenes* calculamos la IOU media para todas las predicciones (haciendo una media aritmética).

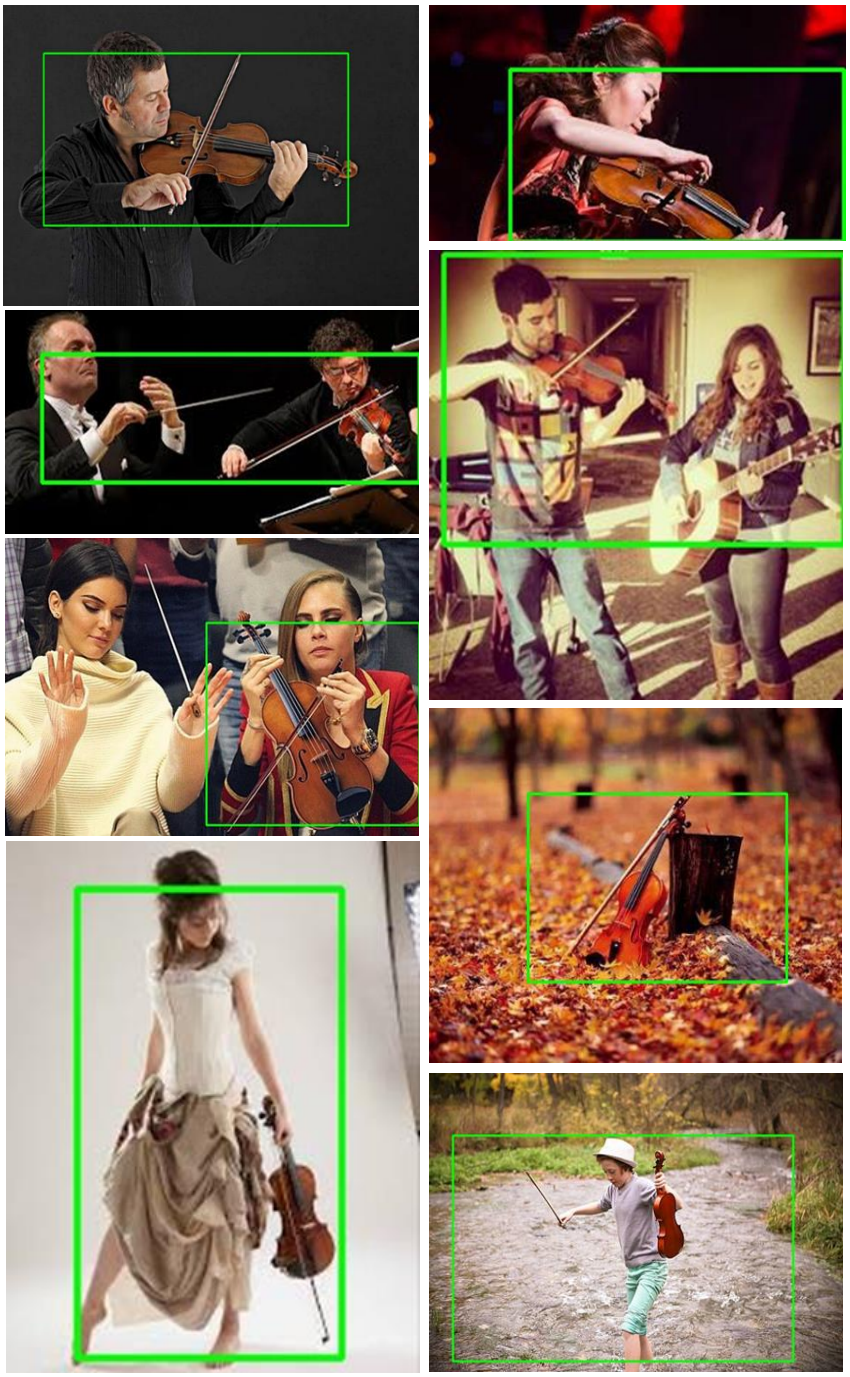
- **Evaluación del modelo construido a partir de mapas de calor**

Dado que la construcción tanto del modelo de Tensorflow como del construido a partir de la segmentación semántica no ha sido exitosa y los resultados eran erróneos ya a simple vista, no hemos evaluado dichos modelos en términos de lo explicado anteriormente. Sin embargo, sí lo hemos hecho con el modelo construido a partir del clasificador de Google usando mapas de calor.

Para ello, hemos tenido en cuenta el valor dado a la función de thresholding (como ya hemos indicado en la sección anterior) y lo hemos ido modificando con el objetivo de poder analizar mejor los resultados. Hemos realizado las pruebas con 40 imágenes del conjunto de imágenes inicial, seleccionadas de tal manera que cumplieran las características necesarias (el violín es el objeto principal, etc). La siguiente tabla muestra los resultados obtenidos:

Valor de thresholding	Precisión obtenida
10	26,42 %
15	31,36 %
20	33,52 %
25	34,21 %
30	34,75 %
40	30,13 %
50	25,50 %
60	19,36 %

Como podemos ver, en el mejor de los casos la precisión obtenida es menor de un 35%, lo que significa que estamos lejos aún de unos buenos resultados. A pesar de variar los valores del thresholding, los resultados no varían mucho, siendo en el valor 30 donde obtenemos una precisión ligeramente mayor. En realidad, si analizamos visualmente las imágenes obtenidas los resultados no parecen malos. Podemos ver algunas de las mismas en las imágenes 44 a 51.



Imágenes 44 a 51: algunos de los resultados del modelo basado en mapas de calor.

Como podemos observar, el principal problema es que en algunas de las imágenes el rectángulo de interés es tan grande que, aunque contiene el violín, también contiene gran parte de la imagen que no tiene interés. Estas imágenes bajan considerablemente el valor de precisión obtenido. En estos casos observamos que ajustar el valor de thresholding no soluciona el problema, así que, en

principio, no podemos hacer nada. Una posible solución futura podría ser fijarnos en la clase a la que pertenece el objeto según el modelo, y, si fuera distinta del violín, descartar la imagen.

Además de medir la precisión del modelo en términos de la IOU, utilizamos otra medida presentada en la fase de diseño que, siendo menos precisa, podrá indicarnos si el modelo se está fijando en el objeto correcto o no. Esta medida consiste en ver si al menos el 50% del área predicha está contenida en el área que contiene al violín realmente. Para implementarlo, añadimos otra función (ver la función *obtener_precision_2* del Apéndice III: Código) que devolverá 1 en caso de que sea así, y 0 en caso de que no. Los resultados obtenidos son los siguientes:

Valor de thresholding	Precisión obtenida
12	17,95 %
40	46,15 %
45	46,15 %
50	56,41 %
52	56,41 %
55	56,41 %
58	51,28 %

Valores de thresholding más altos daban ya precisiones más bajas, puesto que se veía cómo aumentaba el número de imágenes en las que tras aplicar el threshold toda la imagen quedaba igual. Como vemos, la precisión es bastante más alta que antes para valores en torno a 50 y 55. Si bajamos el porcentaje de área que debe estar contenida en el área real a un 40%, obtenemos los siguientes resultados:

Valor de thresholding	Precisión obtenida
12	25,64 %
30	48,71 %
40	61,54 %
42	58,97 %
45	64,10 %
50	64,10 %
60	43,59 %

Como era de esperar, los resultados mejoran ligeramente y llegamos a una precisión del 64%. El problema es que las predicciones no son ajustadas (en muchas de las imágenes el modelo localiza al violín en una región mucho más amplia que la que ocupa el propio violín), o bien otros objetos desvían la atención del modelo y este solo se comporta de la manera esperada para imágenes muy claras.

Si comparamos las precisiones obtenidas en función de las 3 medidas utilizadas, nos quedamos con un valor de threshold en torno a 40 como la mejor opción. Las imágenes 52 a 57 muestran algunos de los resultados obtenidos para dicho valor.

Imágenes 52 a 57: Resultados del modelo basado en mapas de calor para un valor de thresholding de 40.

REFLEXIONES

Vamos a dedicar una breve sección a resumir el proceso seguido y sus motivaciones, con el fin de facilitar al lector la comprensión del mismo.

El problema inicial consistía en construir un modelo capaz de detectar la posición de un violín en una imagen. La manera más directa de abordar este tipo de problemas era la de utilizar una librería de detección de objetos basada en el Aprendizaje Profundo, y escogimos Tensorflow por ser la más extendida y documentada en este ámbito. Sin embargo, si no conseguíamos construir un modelo a partir de esta opción, no descartamos otras posibles alternativas que, aunque menos directas, podían ofrecernos una salida. Las alternativas eran tres: entrenar un modelo que, aunque estaba pensado inicialmente para el problema de la clasificación, aprendiera sobre los puntos que localizan el violín en una imagen; utilizar un modelo de clasificación basado en mapas de calor para localizar el objeto principal (un violín) a partir de los mismos; y utilizar la segmentación semántica para localizar el violín.

Comenzamos con uno de los modelos basados en Faster R-CNN que ofrece Tensorflow, pero el modelo no encuentra el último checkpoint, lo cual nos impidió hacer uso de él, por lo que pasamos a los modelos SSD.

Conseguimos hacer funcionar (entrenar y probar), aparentemente, el modelo ssd_mobilenet_v1_coco, pero los resultados no eran correctos, y no supimos encontrar el error. Dedujimos que la librería era algo inestable y poco documentada, lo que nos impedía seguir por esta vía.

Así, decidimos cambiar de rumbo y optar con una de las alternativas planteadas. Sin embargo, la segunda de ellas, la del modelo que podíamos entrenar para aprender acerca de las coordenadas del rectángulo que localizaba al violín, conllevaba el entrenamiento de una red y, a su vez, incertidumbre en los resultados que obtendríamos. Además, podía suponer una dedicación de tiempo mayor del que disponíamos. Por todo ello, optamos por la siguiente opción: los mapas de calor.

Con los mapas de calor conseguimos localizar los violines, pero los resultados eran bastante imprecisos, y no conseguimos una precisión mayor a un 65%. En este caso, el modelo detectaba más de un objeto principal, o la región que escogía era demasiado amplia (aunque contenía al violín). La única solución que se nos ocurría era que, ya que el modelo estaba pensado para clasificar objetos, podíamos rechazar las imágenes en las que la etiqueta que daba era distinta a la de *violín*. Sin embargo, no llegamos a ejecutar esta idea, sino que decidimos probar con la última de las alternativas para ver si los resultados eran mejores: la segmentación semántica.

Con la segmentación semántica los resultados no fueron mejores. El modelo construido detectaba varios objetos en algunas imágenes, y ninguno de ellos era el violín completo. Por tanto, no daría resultado modificar el modelo para que se quedara con solo uno de ellos (podríamos optar por el mayor). No evaluamos este modelo porque los resultados obtenidos eran, a simple vista, bastante peores que con los mapas de calor.

SEGUIMIENTO Y CONTROL

A pesar de realizar la planificación del proyecto, el desarrollo ha diferido de la misma en algunas ocasiones. Esta sección está dedicada a uno de los puntos más importantes en el desarrollo de proyectos: el seguimiento y control.

El número de horas planificadas era de 300, las cuales se repartían en 5 etapas marcadas por los diversos puntos de control:

La primera etapa abarcaba el periodo entre el inicio del TFG y el primer punto de control, el 15 de octubre; la segunda entre el primer y segundo punto de control, del 15 de octubre al 15 de noviembre; la tercera entre el segundo y tercer punto de control, del 15 de noviembre al 15 de diciembre; la cuarta, entre el tercer y cuarto punto de control, del 15 de diciembre al 15 de enero; y la quinta y última, del 15 de enero hasta el final del TFG, el 18 de febrero.

En la planificación, las 300 horas se distribuían uniformemente en las 5 etapas: 60 horas dedicadas aproximadamente en cada una de ellas. Sin embargo, la realidad ha diferido de la misma.

Siguiendo el diagrama de Gantt de la planificación, las horas dedicadas se distribuían entre las diferentes fases del proyecto como sigue:

- Planificación: 20 horas
- Análisis: 10 horas
- Diseño: 50 horas
- Implementación: 130 horas
- Evaluación: 10 horas
- Conclusiones: 10 horas
- Memoria: 50 horas
- Seguimiento y control: 20 horas

Teniendo en cuenta el mismo diagrama, las tareas a realizar en cada una de las 5 etapas eran las siguientes:

- Etapa 1 (semanas 1 – 6): Planificación, análisis y diseño.
- Etapa 2 (semanas 7 – 11): Diseño e implementación.
- Etapa 3 (semanas 11 – 15): Implementación y evaluación.
- Etapa 4 (semanas 15 – 19): Implementación y evaluación.
- Etapa 5 (semanas 20 – 24): Conclusiones.

Además, en todas ellas también se llevarían a cabo la redacción de la **memoria** y el **seguimiento y control**. El gráfico 1 muestra el reparto de las horas planificado en cada una de las etapas y para cada una de las fases del proyecto de manera gráfica.

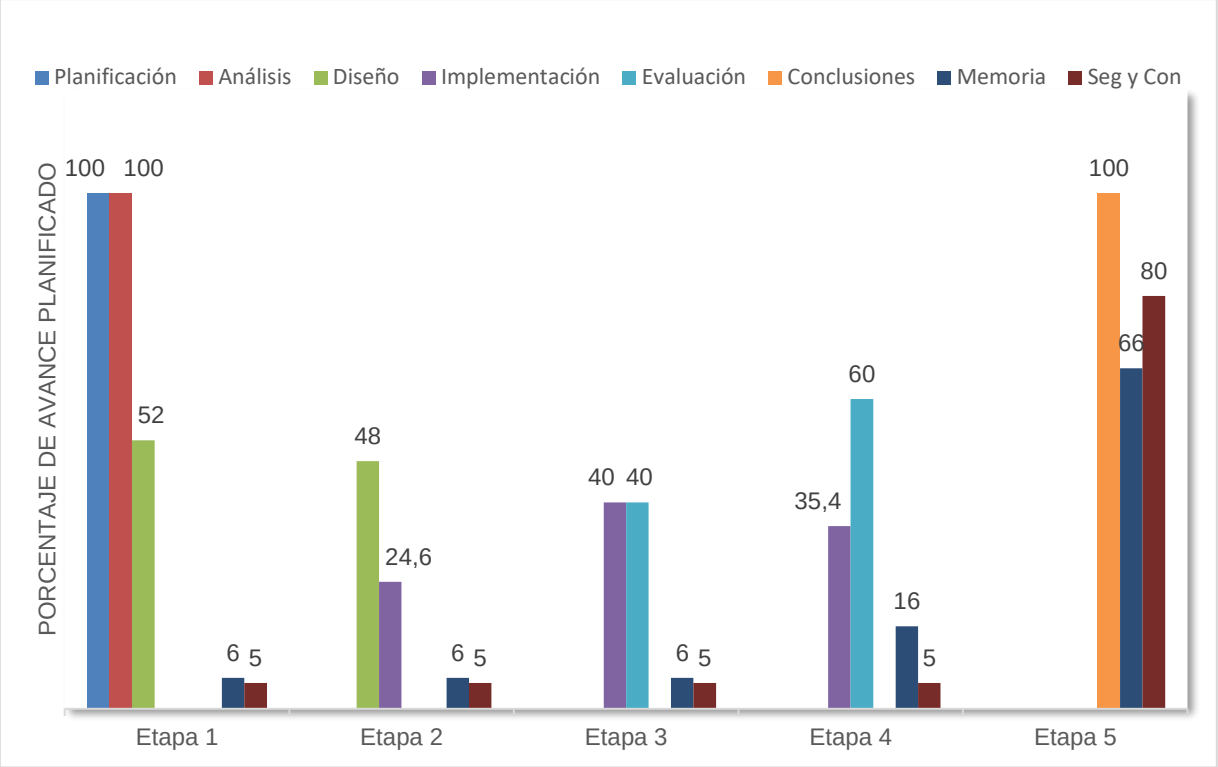


Gráfico 1: Planificación del avance de cada una de las fases del proyecto en las distintas etapas del mismo.

Sin embargo, como ocurre frecuentemente, la realidad no se ajusta del todo a la planificación, sino que difiere en algunos puntos. El gráfico 2 muestra el avance (aproximado) real de las fases a lo largo de las etapas del proyecto.

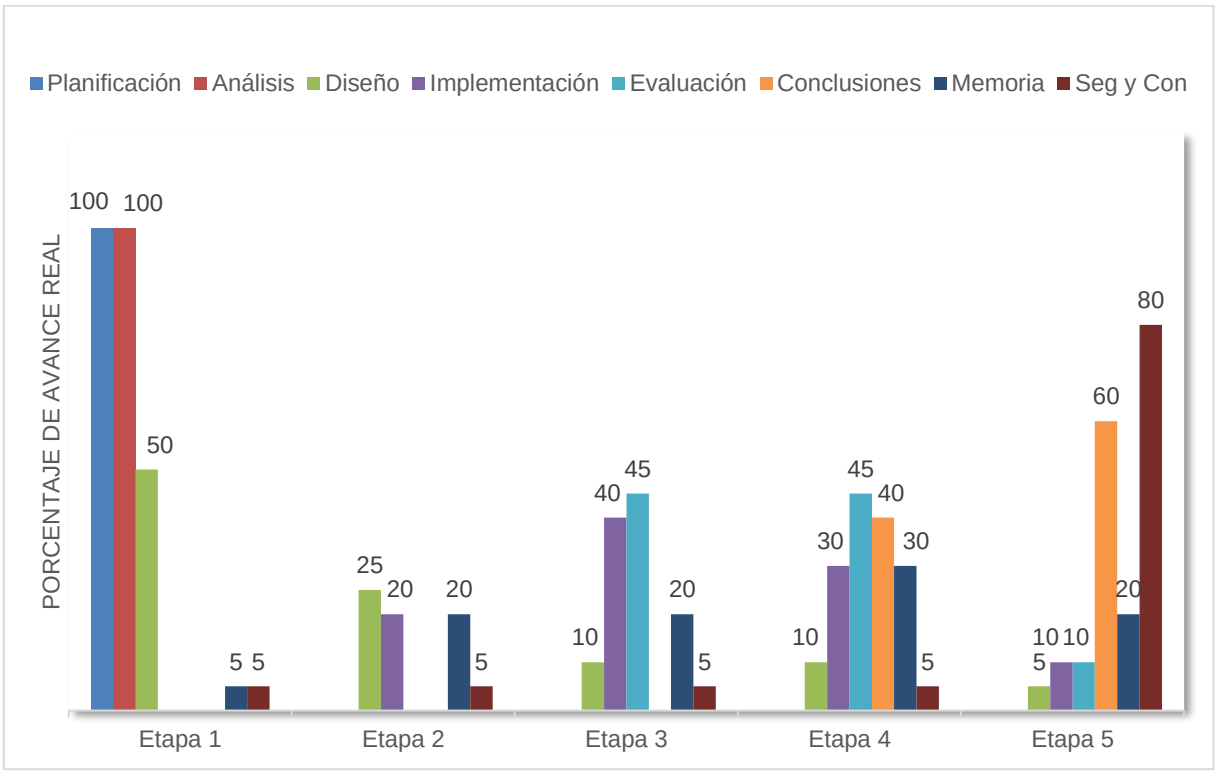


Gráfico 2: Porcentaje real de avance de cada una de las fases del proyecto en las distintas etapas del mismo.

Debemos comentar que el seguimiento y control se ha realizado sobre el progreso, y no sobre las horas exactas, ya que contabilizar las mismas resultaba muy difícil: en muchas ocasiones el ordenador se ha quedado trabajando (entrenando el modelo) y requería de atención intermitente, pero no completa; estas horas no se podían dedicar a otras tareas porque el ordenador quedaba prácticamente inservible… Así, si comparamos el avance real con el planificado podemos observar lo siguiente: el avance de la memoria se ha repartido durante las distintas etapas, y no se ha dejado tanto para la última de ellas como estaba planificado; el diseño debería haber estado acabado para la tercera etapa y, sin embargo, no se ha acabado hasta la última, en la que se han ultimado algunos detalles; y la implementación ha seguido, más o menos, el plan inicial.

Teniendo en cuenta el porcentaje de avance real de cada una de las fases a lo largo del proyecto podemos deducir la relación entre las horas de dedicación planificadas y las dedicadas en cada etapa:

Etapa	Horas planificadas	Horas dedicadas	Desvío
Etapa 1	60	58,5	2,5 %
Etapa 2	60	49,5	17,5 %
Etapa 3	60	72,5	20,83 %
Etapa 4	60	68,5	14,17 %
Etapa 5	60	51	15 %

Lo que nos muestra la tabla es que, sobre todo en la tercera etapa, el trabajo planificado y el real difiere bastante. Esto sucede debido a que en las etapas previas el trabajo realizado está por debajo del planificado en cuanto a avance, lo que produce un mayor esfuerzo en la tercera etapa. Sin embargo, los desvíos no superan en ningún caso el 50%, así que no las consideramos alarmantes. Podemos ver la misma información de manera más visual en el gráfico 3.

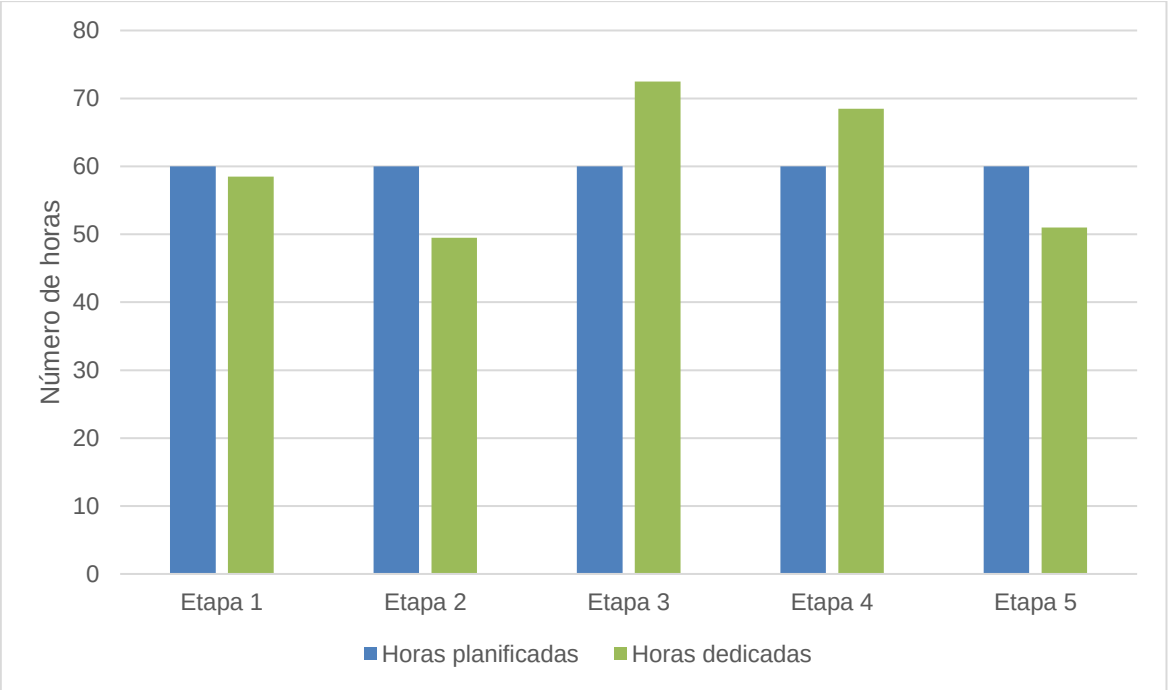


Gráfico 3: Relación entre las horas planificadas y las reales para cada una de las etapas del proyecto.

CONCLUSIONES

La localización y, en general, el mundo de la visión por computador son aún tecnologías poco exploradas y, sobre todo, poco documentadas. El proceso seguido para la construcción de un programa de localización ha sido, en muchos casos, intuitivo, y los resultados obtenidos, muy lejanos a los esperados. La librería de Tensorflow varía constantemente y, de hecho, se encuentran bugs continuamente, lo que entorpece el proceso de manera notoria. Trabajar con una librería en proceso de prueba y reciente conlleva ciertos riesgos, entre los cuales están los que hemos experimentado en este proyecto. Todo este tipo de técnicas que aparentemente son capaces de resolver casi cualquier problema no son ni mucho menos fáciles de usar, y adaptarlas a problemas concretos requiere de mucho trabajo, ya que no están preparadas para usarse directamente.

Una de las conclusiones más importantes del proyecto es que no tenemos que empeñarnos en seguir la vía planificada cuando nos encontramos ante un problema al que no conseguimos dar solución, sino que es importante saber estudiar y plantear nuevas alternativas, aunque en principio cueste dar por perdido el tiempo dedicado. Además, al abordar problemas de este tipo, debemos tener en cuenta que es muy posible que no lleguemos a los resultados esperados o deseados, si no que nos quedemos a medio camino. Una de las posibles ampliaciones de este proyecto, planteada en la sección de requisitos, era la de obtener una alta precisión, superior a un 95%, y, sin embargo, la precisión obtenida no ha alcanzado un 65%. Por tanto, podemos deducir de aquí que, al menos en el problema de la localización, una buena precisión es difícil de obtener (al menos en los términos en los que la hemos planteado).

Aunque no hemos logrado obtener los resultados deseados, una posible labor futura podría ayudar a conseguirlos. Podrían analizarse otras librerías de visión por computador distintas a Tensorflow que utilicen Deep Learning para intentar, utilizando los mismos o similares algoritmos, obtener resultados más satisfactorios.

Personalmente, el proyecto ha supuesto un gran aprendizaje, no solo en el ámbito de la inteligencia artificial y, en concreto, de la visión por computador, sino también sobre la manera de abordar un problema desde varias perspectivas, para construir una solución propia que puede no ser única. Me ha enseñado a no conformarme con la primera opción que creamos que puede servirnos y realizar una búsqueda y una labor de investigación previa. He utilizado muchas librerías de tratamiento de imágenes, de aprendizaje automático y Deep Learning; he investigado sobre las opciones existentes para la anotación de imágenes, y sobre las distintas alternativas (directas e indirectas) para abordar el problema de la localización desde diferentes puntos de vista; he trabajado con Linux y Python, con los que pocas veces me había tocado trabajar; y, sobre todo, he vivido la dificultad de implementar con éxito la inteligencia artificial.

BIBLIOGRAFÍA

[1] G. Mata, «Procesamiento de imágenes biomédicas para el estudio de tratamientos en enfermedades neurodegenerativas. PhD Thesis.,» Universidad de La Rioja, La Rioja, 2017.

[2] J. Heras et al., «GelJ--a tool for analyzing DNA fingerprint gel images.,» *BMC Bioinformatics,* vol. 16, nº 270, 2015.

[3] A. Alonso et al., «AntibiogramJ-a tool for analyzing images from disk diffusion tests.,» *Computer Methods and Programs in Biomedicine,* vol. 143, pp. 159-169, 2017.

[4] I. Goodfellow, Y. Bengio y A. Courville, Deep Learning, MIT Press, 2016.

[5] J. Rey, «tryolabs.com,» 30 Aug 2017. [En línea]. Available: https://tryolabs.com/blog/2017/08/30/object-detection-an-overview-in-the-age-of-deep-learning/?utm_campaign=Revue%20newsletter&utm_medium=Newsletter&utm_source=The%20Wild%20Week%20in%20AI. [Último acceso: Sept 2017].

[6] K. He, X. Zhang, S. Ren and J. Sun, “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification,” in *The IEEE International Conference on Computer Vision (ICCV)*, 2015.

[7] Shah, O. Javed and Mubarak, “Tracking And Object Classification For Automated Surveillance,” University of Central Florida, 4000 Central Florida Blvd, Orlando, Florida 32816, USA.

[8] C. Asplund, «Object classification and localization using machine learning techniques. Designing and training models for use in limited hardware-applications.,» Chalmers University of Technology, Gothenburg, Sweden, 2016.

[9] R. L. White, «Object Classification as a Data Analysis Tool,» *ASP Conf. Ser., Vol. 216, Astronomical Data Analysis Software and Systems IX, eds. N. Manset, C. Veillet, D. Crabtree, 577,* 2000.

[10] H. Jang, H.-J. Yang, D.-S. Jeong y H. Lee, «Object classification using CNN for video traffic detection system,» de *Frontiers of Computer Vision (FCV), 21st Korea-Japan Joint Workshop on Frontiers of Computer Vision (FCV)*, Mokpo, South Korea, 28-30 Jan. 2015.

[11] A. Morales, T. Asfour, P. Azad, S. Knoop y R. Dillmann, «Integrated Grasp Planning and Visual Object Localization For a Humanoid Robot with Five-Fingered Hands,» Beijing, China, 9-15 Oct. 2006.

[12] S. Bag, *Deep Learning Localization for Self-driving Cars,* Rochester: Rochester Institute of Technology, 2017.

[13] E. Ackerman, «Mayfield Robotics Announces Kuri, a $700 Home Robot,» 3 Jan 2017. [En línea].

[14] M. A. Marchetti et al., «Results of the 2016 International Skin Imaging Collaboration International Symposium on Biomedical Imaging challenge: Comparison of the accuracy of computer algorithms to dermatologists for the diagnosis of melanoma from dermoscopic images,» *Journal of the American Academy of Dermatology,* 2017.

[15] A. Staff, «appleinsider,» 4 oct 2017. [En línea]. Available: appleinsider.com. [Último acceso: oct 2017].

[16] «TensorFlow,» [En línea]. Available: www.tensorflow.org. [Último acceso: sept 2017].

[17] G. Oberoi, «goberoi,» 11 Jul 2016. [En línea]. Available: goberoi.com. [Último acceso: sept 2017].

[18] D. Tran, «medium.com,» 28 Jul 2017. [En línea]. Available: https://medium.com/towards-data-science/how-to-train-your-own-object-detector-with-tensorflows-object-detector-api-bec72ecfe1d9. [Último acceso: Sept 2017].

[19] A. Smola y S. V. N. Vishwanathan, Introduction to Machine Learning, Cambrigde University Press, 2010.

[20] S. J. Russel y P. Norvig, Artificial Intelligence: A modern Approach, Pretince Hall, 2003, pp. 1-52.

[21] T. O. Ayodele, «Types of Machine Learning Algorithms,» de *New Advances in Machine Learning*, Yagang Zhang, 2010, pp. 19-48.

[22] Google, «Google imágenes,» [En línea]. Available: https://www.google.es/imghp?hl=es.

[23] «pixelabs,» [En línea]. Available: http://www.pixelabs.es. [Último acceso: 2017].

[24] «instagram,» [En línea]. Available: https://www..com/?hl=es. [Último acceso: 2017].

[25] «The Pascal Visual Object Classes Homepage,» [En línea]. Available: http://host.robots.ox.ac.uk/pascal/VOC/. [Último acceso: Febrero 2018].

[26] «Alp's Labeling Tools for Deep Learning,» [En línea]. Available: https://alpslabel.wordpress.com/. [Último acceso: Octubre 2017].

[27] A. Geiger, P. Lenz, C. Stiller y R. Urtasun, «Vision meets Robotics: The KITTI Dataset,» *International Journal of Robotics Research,* vol. 32, pp. 1229-1235, 2013.

[28] «Annotorious,» [En línea]. Available: https://annotorious.github.io/. [Último acceso: Octubre 2017].

[29] «FastAnnotationTool,» [En línea]. Available: https://github.com/christopher5106/FastAnnotationTool. [Último acceso: Octubre 2017].

[30] «labelImg,» [En línea]. Available: https://github.com/tzutalin/labelImg. [Último acceso: Octubre 2017].

[31] «LabelMe,» [En línea]. Available: http://labelme.csail.mit.edu/Release3.0/. [Último acceso: Octubre 2017].

[32] A. Kläser, «Lear,» [En línea]. Available: https://lear.inrialpes.fr/people/klaeser/software_image_annotation. [Último acceso: Octubre 2017].

[33] ryouchinsa, «Rectlabel-support,» [En línea]. Available: https://github.com/ryouchinsa/Rectlabel-support. [Último acceso: noviembre 2017].

[34] A. Dutta, A. Gupta y A. Zisserman, «Visual Geoetry Group,» Department of Engineering Science, University of Oxford, [En línea]. Available: http://www.robots.ox.ac.uk/~vgg/software/via/. [Último acceso: noviembre 2017].

[35] wiany11, «jsoda,» [En línea]. Available: https://github.com/wiany11/jsoda. [Último acceso: noviembre 2017].

[36] «Philosys Software GMBH,» [En línea]. Available: https://www.philosys.de/en/services/annotation-services. [Último acceso: noviembre 2017].

[37] sgp715, «simple_image_annotator,» [En línea]. Available: https://github.com/sgp715/simple_image_annotator. [Último acceso: noviembre 2017].

[38] A. S. Razavian et al., «CNN features off-the-shelf: An astounding baseline for recognition,» de *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW'14)*, 2014.

[39] «COCO: Common Objects in Context,» [En línea]. Available: cocodataset.org. [Último acceso: diciembre 2017].

[40] A. Geiger, P. Lenz, C. Stiller y R. Urtasun, «The KITTI Vision Benchmark Suite,» Karlsruhe Institute of Technology and Toyota Tecnhological Institute at Chicago, [En línea]. Available: http://www.cvlibs.net/datasets/kitti/. [Último acceso: diciembre 2017].

[41] openimages, «The Open Images dataset,» [En línea]. Available: https://github.com/openimages/dataset. [Último acceso: diciembre 2017].

[42] S. Ren, K. He, R. Girshick y J. Sun, *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,* Cornell University Library, 2016.

[43] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Fu y A. C. Berg, *SSD: Single Shot MultiBox Detector,* Cornell University Library, 2016.

[44] «Keras: The Python Deep Learning library,» [En línea]. Available: https://keras.io/. [Último acceso: octubre 2017].

[45] facebookresearch, «Detectron,» [En línea]. Available: https://github.com/facebookresearch/Detectron. [Último acceso: octubre 2017].

[46] S. J. Pan y Q. Yang, «A Survey on Transfer Learning,» *IEEE transactions on knowledge and data engineering,* pp. 1-15, 2009.

[47] A. Cook, «Global Average Pooling Layers for Object Localization,» 9 Abril 2017. [En línea]. Available: https://alexisbcook.github.io. [Último acceso: Diciembre 2017].

[48] A. Rosebrock, «pyimagesearch,» 7 noviembre 2016. [En línea]. Available: https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/. [Último acceso: noviembre 2017].

[49] K. He, X. Zhang, S. Ren y J. Sun, «Deep Residual Learning for Image Recognition,» *Cornell University Library,* 2015.

[50] E. N. Kaur y E. Y. Kaur, «Object classification Techniques using Machine Learning Model,» *International Journal of Computer Trends and Technology (IJCTT),* pp. 170 - 174, Dec 2014.

APÉNDICE I: TIPOS DE ALGORITMOS DE APRENDIZAJE AUTOMÁTICO

Los diferentes algoritmos de Aprendizaje Automático se agrupan en una taxonomía en función de la salida de los mismos. Algunos tipos de algoritmos son los siguientes:

Los algoritmos de **aprendizaje supervisado** producen una función que establece una correspondencia entre las entradas y las salidas deseadas del sistema. La base de conocimiento del sistema está formada por ejemplos de etiquetados anteriores.

En **el aprendizaje no supervisado** todo el proceso de modelado se lleva a cabo sobre un conjunto de ejemplos formado tan sólo por entradas al sistema. No se tiene información sobre las categorías de esos ejemplos. Por lo tanto, en este caso, el sistema tiene que ser capaz de reconocer patrones para poder etiquetar las nuevas entradas.

El **aprendizaje semi-supervisado** utiliza datos de entrenamiento tanto etiquetados como no etiquetados (normalmente una pequeña cantidad de datos etiquetados junto a una gran cantidad de datos no etiquetados).

El **aprendizaje por refuerzo** se inspira en la psicología conductista, cuya ocupación es determinar qué debe hacerse en un entorno dado con el fin de maximizar alguna noción de “recompensa”.

La **transducción** es similar al aprendizaje supervisado, pero no construye una función de forma explícita, sino que trata de predecir el resultado correcto de los próximos datos de entrada basándose en los ejemplos de entrada.

En **el aprendizaje multi-tarea** se resuelven múltiples tareas de aprendizaje al mismo tiempo aprovechando las similitudes y las diferencias entre tareas, lo que puede conllevar una mayor eficiencia en el aprendizaje y precisión predictiva en los modelos específicos de la tarea, en comparación con las capacidades de los modelos por separado.

APÉNDICE II: TIPOS DE TRANSFORMACIONES

El modelo **HSV** (Hue, Saturation, Value), también llamado HSB (Hue, Saturation, Brightness), define un modelo de color en términos de sus componentes. En él, el matiz se representa por una región circular; una región triangular separada puede ser usada para representar la saturación y el valor del color.

LAB es el nombre abreviado de dos espacios de color diferentes: CIELAB y Lab. El propósito de ambos espacios es producir un espacio de color más “perceptivamente lineal” que otros espacios de color.

La **ecualización adaptativa del histograma** (AHE) es una técnica de procesamiento de imágenes por computador que se utiliza para mejorar el contraste en las imágenes. Difiere de la ecualización de histograma ordinaria en el sentido de que el método adaptativo computa varios histogramas, cada uno correspondiente a una sección distinta de la imagen, y los utiliza para redistribuir los valores de luminosidad de la imagen.

El **ruido sal y pimienta** (salt-and-pepper noise) se caracteriza principalmente por cubrir de forma dispersa toda la imagen con una serie de píxeles blancos y negros.

La **erosión** es un tipo de transformación morfológica básica. No entraremos más en detalle porque se trata de una transformación más teórica que las demás.

El **desenfoque Gaussiano** es un efecto de suavizado para imágenes. Esencialmente, el efecto mezcla ligeramente los colores de los píxeles que estén vecinos el uno al otro en una imagen, lo que provoca que la imagen pierda algunos detalles minúsculos y, de esta forma, hace que la imagen se vea más suave (aunque menos nítida o clara). Se genera un efecto similar al de una fotografía tomada con una cámara fotográfica desenfocada.

La **posterización** o cartelización es la conversión o reproducción de una imagen de tonos continuos para obtener otra imagen en la que sólo hay unos pocos tonos diferenciados y presentando una calidad tipo póster.

APÉNDICE III: CÓDIGO

funciones_auxiliares.py

```
import xml.etree.cElementTree as ET
import cv2
from lxml import etree
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import os

source_xml = "imagenes_xml_inicial/"

# La siguiente funcion devuelve las coordenadas del rectangulo blanco dada una imagen negra que
contiene un rectangulo blanco horizontal
def obtener_rectangulo(image):
    width, height = image.shape
    for x in range (0, width ):
        for y in range (0, height):
            if image[x,y]==255:
                for y2 in range (y, height):
                    if image[x,y2]==0:
                        for x2 in range (x, width):
                            if image[x2,y]==0:
                                return (y+1,x+1,y2,x2)

def generar_y_guardar_xml(imagen_transformada, carpetaOrigen, rutaCompleta, nomImagen ,
(x0,y0,x1,y1), destiny_xml, nomXml):

    doc = ET.Element("annotation")

    ET.SubElement(doc, "folder").text = carpetaOrigen
    ET.SubElement(doc, "filename").text = nomImagen
    ET.SubElement(doc, "path").text = rutaCompleta
    src = ET.SubElement(doc, "source")
    ET.SubElement(src, "database").text = "Unknown"
    size = ET.SubElement(doc, "size")
    ET.SubElement(size, "width").text = str(imagen_transformada.shape[1])
    ET.SubElement(size, "height").text = str(imagen_transformada.shape[0])
    if(len(imagen_transformada.shape)==3):
        ET.SubElement(size, "depth").text = "3"
    else:
        ET.SubElement(size, "depth").text = "1"

    ET.SubElement(doc, "segmented").text = "0"
    object = ET.SubElement(doc, "object")
    ET.SubElement(object, "name").text = "violin"
    ET.SubElement(object, "pose").text = "Unspecified"
    ET.SubElement(object, "truncated").text = "0"
    ET.SubElement(object, "difficult").text = "0"

    bndbox = ET.SubElement(object, "bndbox")
    ET.SubElement(bndbox, "xmin").text = str(x0)
    ET.SubElement(bndbox, "ymin").text = str(y0)
    ET.SubElement(bndbox, "xmax").text = str(x1)
    ET.SubElement(bndbox, "ymax").text = str(y1)
```

```
    tree = ET.ElementTree(doc)
    tree.write(destiny_xml +"/"+ nomXml, xml_declaration=True, method='xml')

def extraer_imagen_anotacion_desde_xml(fichero):

    doc = etree.parse(source_xml +fichero)

    nombreImagen= doc.find('folder').text + "/" + doc.find('filename').text

    (x0, y0, x1, y1) = (doc.find('object').find('bndbox').find('xmin').text,
                        doc.find('object').find('bndbox').find('ymin').text,
                        doc.find('object').find('bndbox').find('xmax').text,
                        doc.find('object').find('bndbox').find('ymax').text)

    imagenAux = cv2.imread(nombreImagen, 3)
    (b, g, r) = cv2.split(imagenAux)
    imagen = cv2.merge([r, g, b])
    return imagen, nombreImagen, (int(x0),int(y0),int(x1),int(y1))

def visualizar_imagen_anotada(xml):
    doc = etree.parse(xml)

    nombreImagen = doc.find('folder').text + "/" + doc.find('filename').text

    (x0, y0, x1, y1) = (doc.find('object').find('bndbox').find('xmin').text,
                        doc.find('object').find('bndbox').find('ymin').text,
                        doc.find('object').find('bndbox').find('xmax').text,
                        doc.find('object').find('bndbox').find('ymax').text)
    imagen = mpimg.imread(nombreImagen, 0)
    imagen = cv2.rectangle(imagen, (int(x0), int(y0)), (int(x1), int(y1)), (111, 255, 222), 2)
    plt.imshow(imagen)
    plt.show()


def visualizar_imagenes_anotadas(directorio):
    for file in os.listdir(directorio):
        visualizar_imagen_anotada(directorio+"/"+file)


def cambiar_XML_iniciales( source_xml, destiny_xml, nueva_carpeta_imagenes, nueva_ruta_carpeta):

    for file in os.listdir(source_xml):
        docOri = etree.parse(source_xml+"/"+file)

        rutaImagenCompleta = docOri.find('folder').text + "/" + docOri.find('filename').text

        (x0, y0, x1, y1) = (docOri.find('object').find('bndbox').find('xmin').text,
                            docOri.find('object').find('bndbox').find('ymin').text,
                            docOri.find('object').find('bndbox').find('xmax').text,
                            docOri.find('object').find('bndbox').find('ymax').text)
        filename = docOri.find('filename').text
        (shape0, shape1, shape2)  = (docOri.find('size').find('width').text,
docOri.find('size').find('height').text, docOri.find('size').find('depth').text)
        doc = ET.Element("annotation")

        ET.SubElement(doc, "folder").text = nueva_carpeta_imagenes
        ET.SubElement(doc, "filename").text = filename
```

```
        ET.SubElement(doc, "path").text = nueva_ruta_carpeta+"/"+filename
        src = ET.SubElement(doc, "source")
        ET.SubElement(src, "database").text = "Unknown"
        size = ET.SubElement(doc, "size")
        ET.SubElement(size, "width").text = shape0
        ET.SubElement(size, "height").text = shape1
        ET.SubElement(size, "depth").text = shape2

        ET.SubElement(doc, "segmented").text = "0"
        object = ET.SubElement(doc, "object")
        ET.SubElement(object, "name").text = "violin"
        ET.SubElement(object, "pose").text = "Unspecified"
        ET.SubElement(object, "truncated").text = "0"
        ET.SubElement(object, "difficult").text = "0"

        bndbox = ET.SubElement(object, "bndbox")
        ET.SubElement(bndbox, "xmin").text = x0
        ET.SubElement(bndbox, "ymin").text = y0
        ET.SubElement(bndbox, "xmax").text = x1
        ET.SubElement(bndbox, "ymax").text = y1

        tree = ET.ElementTree(doc)
        tree.write(destiny_xml+"/" + file, xml_declaration=True, method='xml')
```

aumentar_dataset.py

```
import os
import scipy.misc
from funciones_tranformar import voltear, salpimentar, quitar_color, girar_90_grados
from funciones_auxiliares import extraer_imagen_anotacion_desde_xml, generar_y_guardar_xml,
visualizar_imagen_anotada

# source_xml sera el directorio que contiene todos los archivos xml
# destiny_xml sera el directorio que contendra los xml generados
# destiny sera el directorio de las nuevas imagenes generadas
source_xml = "imagenes_xml_inicial"
destiny_xml = "prueba"
destiny = "imagenes_aumentado"

# en la siguiente lista metemos todas las funciones transformadoras que queremos aplicar a nuestro
dataset
lista_transformadores = [voltear, quitar_color]
i = 1
for file in os.listdir(source_xml):
    # para cada imagen le aplicamos cada una de las transformaciones y la guardamos
    (imagen, nomImgOriginal, (x0,y0,x1,y1)) = extraer_imagen_anotacion_desde_xml(file)
    for funcion in lista_transformadores:
        (imagen_transformada, (x0_t,y0_t,x1_t,y1_t)) = funcion(imagen, (x0,y0,x1,y1))
        nomImagen = "aux" + `i` + ".jpeg"
        scipy.misc.imsave(destiny+"/"+nomImagen, imagen_transformada)

        rutaCompleta ="/home/lucia/Escritorio/lucia/localizador/"+destiny+"/"+nomImagen
        nomXml = "aux" + `i` + ".xml"
        xml = generar_y_guardar_xml(imagen_transformada, destiny,rutaCompleta, nomImagen , (x0_t,
y0_t, x1_t, y1_t), destiny_xml, nomXml)
        i = i+1
        # si queremos visualizar la imagen anotada creada, descomentamos la siguiente linea
        # visualizar_imagen_anotada(destiny_xml+nomXml)
```

```
exit()
```

funciones_transformar.py

```
import cv2
import numpy as np
from funciones_auxiliares import obtener_rectangulo

# nuestras funciones transformadoras para tener como parametros de entrada: la imagen y la
anotacion original
# (dos puntos), y van a devolver: la imagen transformada, y la nueva anotacion

def voltear(imagen, (x0,y0,x1,y1)):
    img_volteada = cv2.flip(imagen, -1)

    # creamos la imagen negra con el rectangulo blanco, y se la pasamos a la funcion
obtener_rectangulo para que obtenga la nueva anotacion
    img_negra = np.zeros(imagen.shape[:2], dtype="uint8");
    cv2.rectangle(img_negra, (x0, y0), (x1, y1), 255, -1);

    img_negra_volteada = cv2.flip(img_negra, -1);
    (x0_t,y0_t,x1_t,y1_t) = obtener_rectangulo(img_negra_volteada);
    return img_volteada, (x0_t,y0_t,x1_t,y1_t);

def quitar_color(imagen, (x0,y0,x1,y1)):
    gray = rgb2gray(imagen)
    return gray, (x0,y0,x1,y1)

def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.299, 0.587, 0.114])

def salpimentar(imagen, (x0,y0,x1,y1)):
    s_vs_p = 0.5
    amount = 0.004
    imagen_transformada = np.copy(imagen)
    # Sal
    num_salt = np.ceil(amount * imagen.size * s_vs_p)
    coords = [np.random.randint(0, i - 1, int(num_salt))
            for i in imagen.shape]
    imagen_transformada[coords] = 1

    # Pimienta
    num_pepper = np.ceil(amount * imagen.size * (1. - s_vs_p))
    coords = [np.random.randint(0, i - 1, int(num_pepper))
            for i in imagen.shape]
    imagen_transformada[coords] = 0

    return imagen_transformada, (x0,y0,x1,y1)

def girar_90_grados(imagen, (x0,y0,x1,y1)):
    rows, cols = imagen.shape

    M = cv2.getRotationMatrix2D((cols / 2, rows / 2), 90, 1)
    img_girada = cv2.warpAffine(imagen, M, (cols, rows))

    img_negra = np.zeros(imagen.shape[:2], dtype="uint8");
    img_negra_girada = cv2.warpAffine(img_negra, M, (cols, rows))
    cv2.rectangle(img_negra, (x0, y0), (x1, y1), 255, -1);
```

```
    (x0_t, y0_t, x1_t, y1_t) = obtener_rectangulo(img_negra_girada);

    return img_girada, (x0_t, y0_t, x1_t, y1_t)
```

evaluacion.py

```
PATH_TO_TEST_IMAGES_DIR = 'images_planB'

def pretrained_path_to_tensor(img_path):
    # loads RGB image as PIL.Image.Image type
    img = image.load_img(img_path, target_size=(224, 224))
    # convert PIL.Image.Image type to 3D tensor with shape (224, 224, 3)
    x = image.img_to_array(img)
    # convert 3D tensor to 4D tensor with shape (1, 224, 224, 3) and return 4D tensor
    x = np.expand_dims(x, axis=0)
    # convert RGB -> BGR, subtract mean ImageNet pixel, and return 4D tensor
    return preprocess_input(x)

def get_ResNet():
    # define ResNet50 model
    model = ResNet50(weights='imagenet')
    # get AMP layer weights
    all_amp_layer_weights = model.layers[-1].get_weights()[0]
    # extract wanted output
    ResNet_model = Model(inputs=model.input,
        outputs=(model.layers[-4].output, model.layers[-1].output))
    return ResNet_model, all_amp_layer_weights

def ResNet_CAM(img_path, model, all_amp_layer_weights):
    # get filtered images from convolutional output + model prediction vector
    last_conv_output, pred_vec = model.predict(pretrained_path_to_tensor(img_path))
    # change dimensions of last convolutional outpu tto 7 x 7 x 2048
    last_conv_output = np.squeeze(last_conv_output)
    # get model's prediction (number between 0 and 999, inclusive)
    pred = np.argmax(pred_vec)
    # bilinear upsampling to resize each filtered image to size of original image
    mat_for_mult = scipy.ndimage.zoom(last_conv_output, (32, 32, 1), order=1) # dim: 224 x 224 x
2048
    # get AMP layer weights
    amp_layer_weights = all_amp_layer_weights[:, pred] # dim: (2048,)
    # get class activation map for object class that is predicted to be in the image
    final_output = np.dot(mat_for_mult.reshape((224*224, 2048)),
amp_layer_weights).reshape(224,224) # dim: 224 x 224
    # return class activation map
    return final_output, pred


def save_ResNet_CAM(img_path, ax, model, all_amp_layer_weights, file):
    i = file.split(".")[0]
    (xx0, yy0, xx1, yy1, altura, hanchura) = obtenerCoordenadasDeXml(i)
    im = cv2.resize(cv2.cvtColor(cv2.imread(img_path), cv2.COLOR_BGR2RGB), (224, 224))

    CAM, pred = ResNet_CAM(img_path, model, all_amp_layer_weights)
    cv2.imwrite("cam.jpg", CAM)
    (T, thresh) = cv2.threshold(cv2.imread("cam.jpg", 0), 12, 255, cv2.THRESH_BINARY)
    cnts = cv2.findContours(thresh, 1, 2)
```

```
    cnts = cnts[0] if imutils.is_cv2() else cnts[1]
    clone = im.copy()
    os.remove("cam.jpg")

    j = 0
    IOU_SUMA = 0
    c = max(cnts, key=cv2.contourArea)
    (x, y, w, h) = cv2.boundingRect(c)

    # draw the book contour (in green)
    cv2.rectangle(clone, (x, y), (x + w, y + h), (0, 255, 0), 2)

    im_original = cv2.imread(img_path)
    (xx0, yy0, xx1, yy1, altura, hanchura) = obtenerCoordenadasDeXml(i)
    xx0 = int(hanchura * x)/ 224
    xx1 = int(hanchura * (x+w)) / 224
    yy0 = int(altura * y) / 224
    yy1 = int(altura * (y+h)) / 224

    cv2.rectangle(im_original, (xx0, yy0), (xx1, yy1), (0, 255, 0), 2)
    cv2.imwrite("predicciones_planB/prediccion" + file, im_original)
    # IOU = obtener_IOU(i, (xx0, yy0, xx1, yy1))
    IOU = obtener_precision_2(i, (xx0, yy0, xx1, yy1))
    # cv2.imwrite("predicciones_planB/prediccion"+ file, clone)
    return IOU

def predecirImagenes(ax, ResNet_model, all_amp_layer_weights):
    IOU_SUMA = 0
    i = 0.
    for file in os.listdir(PATH_TO_TEST_IMAGES_DIR):
        IOU = save_ResNet_CAM(PATH_TO_TEST_IMAGES_DIR+"/"+file, ax, ResNet_model,
all_amp_layer_weights, file)
        IOU_SUMA = IOU_SUMA + IOU
        i = i + 1
    print ("Precision total: ", IOU_SUMA / i)

if __name__ == '__main__':
    ResNet_model, all_amp_layer_weights = get_ResNet()
    # img_path = sys.argv[1]
    fig, ax = plt.subplots()
    # CAM = plot_ResNet_CAM(img_path, ax, ResNet_model, all_amp_layer_weights)
    predecirImagenes(ax, ResNet_model, all_amp_layer_weights)
    # plt.show()
```

evaluar_IOU_media.py

```
PATH_TO_LABELS = os.path.join('', '../violines_mapa_etiquetas.pbtxt')
NUM_CLASSES = 1
PATH_TO_TEST_IMAGES_DIR = '/home/lucia/Escritorio/lucia/localizador/imagenes_inicial/'
PATH_TO_XML_TEST_DIR = '/home/lucia/Escritorio/lucia/localizador/imagenes_xml_inicial/test/'
TEST_IMAGE_PATHS = [ os.path.join(PATH_TO_TEST_IMAGES_DIR, '{}.jpeg'.format(i)) for i in
range(301,401) ]
PATH_TO_PREDICTED_IMAGES_DIR = "imagenes_test_predichas/"
IMAGE_SIZE = (12, 8)
```

```
label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories = label_map_util.convert_label_map_to_categories(label_map,
max_num_classes=NUM_CLASSES, use_display_name=True)
category_index = label_map_util.create_category_index(categories)

def load_image_into_numpy_array(image):
  (im_width, im_height) = image.size
  return np.array(image.getdata()).reshape(
      (im_height, im_width, 3)).astype(np.uint8)

def obtenerCoordenadasDeXml(i):
    xml = PATH_TO_XML_TEST_DIR+ i+".xml"
    doc = etree.parse(xml)
    (x0, y0, x1, y1) = (doc.find('object').find('bndbox').find('xmin').text,
                        doc.find('object').find('bndbox').find('ymin').text,
                        doc.find('object').find('bndbox').find('xmax').text,
                        doc.find('object').find('bndbox').find('ymax').text)
    (hanchura, altura) = (doc.find('size').find('width').text,
                        doc.find('size').find('height').text)
    return (int(x0),int(y0),int(x1),int(y1), int(altura), int(hanchura))


# (xx0, yy0, xx1, yy1) son las coordenadas predichas para la imagen 'i'.jpeg
def interseccion(i, (xx0, yy0, xx1, yy1)):
    #Obtengo las coordenadas reales
    (x0, y0, x1, y1, altura, hanchura) = obtenerCoordenadasDeXml(i)
    print ("reales", (x0, y0, x1, y1), "predichas", (xx0, yy0, xx1, yy1))
    if (x1 < xx0 or x0 > xx1 or y0 < yy1 or y1 > yy0):
        # interseccionan. Calculamos la interseccion
        dx = min(x1, xx1) - max(x0, xx0)
        dy = min(y1, yy1) - max(y0, yy0)
        return dx * dy

    else:
        # no interseccionan
        return 0

def interseccion2((x0, y0, x1, y1),(xx0, yy0, xx1, yy1), altura, hanchura ):
    dx = min(x1, xx1) - max(x0, xx0)
    dy = min(y1, yy1) - max(y0, yy0)
    return dx * dy

def union(i, (xx0, yy0, xx1, yy1)):
    (x0, y0, x1, y1, altura, hanchura) = obtenerCoordenadasDeXml(i)

    if (x1 < xx0 or x0 > xx1 or y0 < yy1 or y1 > yy0):
        union = ((xx1 - xx0) * (yy1 - yy0) + (x1 - x0) * (y1 - y0)) - interseccion2((x0, y0, x1, y1),
                                                    (xx0, yy0, xx1, yy1), altura,
                                                    hanchura)
        return union
    else:  # interseccionan. Calculamos la interseccion

        # no interseccionan. la union sera la suma de ambas areas
        return (((int(xx1) - int(xx0))) * (int(yy1) - int(yy0)) + ((int(x1) - int(x0)) * (int(y1) - int(y0))))
```

```
def obtener_IOU(i, (x0,y0,x1,y1)):
    return (interseccion(i, (x0,y0,x1,y1))/float(union(i, (x0,y0,x1,y1))))

def obtener_precision _2(i, (x0,y0,x1,y1)):
    # Si el 50 por ciento del area predicha esta contenido en el area real devolvemos un 1, si no, un
0
    if (interseccion(i, (x0,y0,x1,y1)) >= 0.5*((x1 - x0)*(y1 - y0))):
        return 1
    else:
        return 0
```